Paul Chlebek

User Interface-orientierte Softwarearchitektur

- Bauentwurfslehre für interaktive Softwareoberflächen
- Kompass für die Entwicklung dialogintensiver Anwendungen
- Leitfaden für erlebbare User Interfaces

Mit Online-Service zum Buch

vieweg

Paul Chlebek

User Interface-orientierte Softwarearchitektur

Aus dem Bereich IT erfolgreich gestalten

Visual Basic .NET mit Methode
von Heinrich Rottmann

Warum ausgerechnet .NET?
von Heinrich Rottmann

Requirements Analysis realisieren
von Karl Scharbert

Management der Software-Entwicklung
von Carl Steinweg

Das neue PL/I
von Eberhard Sturm

Projektmanagement der SW-Entwicklung
von Werner Mellis

Profikurs ABAP®
von Patrick Theobald

SAP R/3® Kommunikation mit RFC und Visual Basic
von Patrick Theobald

Six Sigma in der SW-Entwicklung
von Thomas Michael Fehlmann

Profikurs Eclipse 3
von Gottfried Wolmeringer und Thorsten Klein

Erfolgreiche Datenbankanwendung mit SQL3
von Jörg Fritze und Jürgen Marsch

Web-basierte Systemintegration
von Harry Marsh Sneed und Stephan Henry Sneed

Terminalserver mit Citrix Metaframe XP
von Thomas Joos

Exchange Server 2000
von Thomas Joos

Profikurs PHP-Nuke
von Jens Ferner

Unternehmensweites Datenmanagement
von Rolf Dippold, Andreas Meier, Walter Schnider und Klaus Schwinn

Netzarchitektur - Kompass für die Realisierung
von Thomas Spitz, Markus Blümle und Holger Wiedel

SIP - Die Technik
von Andreas Kanbach

IT-Sicherheit - Make or Buy
von Marco Kleiner, Lucas Müller und Mario Köhler

Mehr IT-Sicherheit durch Pen-Tests
von Enno Rey, Michael Thumann und Dominick Baier

IT-Risiko-Management mit System
von Hans-Peter Königs

IT-Sicherheit mit System
von Klaus-Rainer Müller

Der IT Security Manager
von Heinrich Kersten und Gerhard Klett

User Interface-orientierte Softwarearchitektur
von Paul Chlebek

www.vieweg.de

Paul Chlebek

User Interface-orientierte Softwarearchitektur

Bauentwurfslehre für interaktive Softwareoberflächen – Kompass für die Entwicklung dialogintensiver Anwendungen – Leitfaden für erlebbare User Interfaces

Mit 85 Abbildungen

vieweg

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Das in diesem Werk enthaltene Programm-Material ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Der Autor übernimmt infolgedessen keine Verantwortung und wird keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne von Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürfen.

Höchste inhaltliche und technische Qualität unserer Produkte ist unser Ziel. Bei der Produktion und Auslieferung unserer Bücher wollen wir die Umwelt schonen: Dieses Buch ist auf säurefreiem und chlorfrei gebleichtem Papier gedruckt. Die Einschweißfolie besteht aus Polyäthylen und damit aus organischen Grundstoffen, die weder bei der Herstellung noch bei der Verbrennung Schadstoffe freisetzen.

1. Auflage August 2006

Alle Rechte vorbehalten
© Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH, Wiesbaden 2006

Lektorat: Günter Schulz / Andrea Broßler

Der Vieweg-Verlag ist ein Unternehmen von Springer Science+Business Media.
www.vieweg.de

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Konzeption und Layout des Umschlags: Ulrike Weigel, www.CorporateDesignGroup.de
Umschlagbild: Nina Faber de.sign, Wiesbaden
Druck- und buchbinderische Verarbeitung: MercedesDruck, Berlin
Printed in Germany

ISBN-10 3-8348-0162-3
ISBN-13 978-3-8348-0162-3

User Interface-Architektur

- Anleitung zur User Interface-orientierten Softwarearchitektur
- Bauentwurfslehre für durchdachte interaktive Softwareoberflächen
- Kompass für die benutzerzentrierte Entwicklung dialogintensiver Anwendungen
- Leitfaden für erlebbare und eindeutige User Interface-Spezifikationen

Später für Léon; jetzt für Markus.

Paul Chlebek

Über den Autor

Paul Chlebek ist Experte für UML (Unified Modelling Language) und XML (Extensible Markup Language). Der Schwerpunkt seiner derzeitigen Projekt- und Forschungsarbeit liegt im Entwickeln von Methoden und domänenspezifischen Sprachen für die Konstruktion von Softwareoberflächen.

Durch Verbindung der Spezialgebiete HCI (Mensch Maschine Schnittstellen), DSL (domänenspezifische Sprachen) und MDA (modellgetriebene Architektur) hat er die die User Interface-orientierte Softwarearchitektur aufgestellt.

Er bringt ein breites Spektrum an Erfahrungen aus der Projektpraxis im In- und Ausland in den Rollen Leitender Entwickler, Internationalisierungsingenieur, Projektleiter, Technischer Entwicklungsleiter, Softwarearchitekt und Testmanager mit.

Paul Chlebek arbeitet freiberuflich als Berater und Methodenentwickler für BMW und andere international tätige Unternehmen. Er ist Mitglied in der Gesellschaft für Informatik, in der International Association of Software Architects und im Berufsverband der deutschen Usability Professionals sowie Gründer des Open HCI Knowledge Projekts www.BenutzerOberflaeche.de.

Autorenhomepage

https://www.openbc.com/hp/Paul_Chlebek/

http://www.projectpeople.net/chlebek/

http://www.benutzeroberflaeche.de

Geleitwort

Geleitwort des Praxismentors

Dialoganforderungen werden komplexer

Als die Welt noch keine Dialog-Software kannte, waren die Anstrengungen von Software-Entwicklern auf die funktionale Seite und die Optimierung von Rechenzentrum-Elementen wie einfaches Operating und Datenspeicher-Minimierung gerichtet. Der Benutzer hatte mit der resultierenden Anwendung im Regelfall keine direkte Berührung gehabt und nur deren Resultate oder Wirkungen wahrgenommen. Die Entwicklung der Anwendung dauerte ihre Zeit, aber man lebte damit.

Nach der ersten Dialogisierung war eine interessante Feststellung zu machen: In dem Maße, in dem die Softwareentwicklung komplexer wurde, eine höhere Fertigungstiefe erhielt und Fehler oder konzeptionelle Missinterpretationen sofort sichtbar wurden, stiegen die Anforderungen an die Umsetzungszeit.

Je intensiver also die Benutzer Einfluss auf die Gestaltung nehmen konnten, umso schneller wollten sie Ergebnisse sehen, obwohl der Entwicklungsaufwand zunahm.

Diese Entwicklung ist bis heute nicht abgeschlossen, trotz (oder wegen) agilem Manifest, SOA und vielen anderen Technologie-Parametern.

Die Ausführungen des Autors zur User Interface-orientierten Architektur helfen uns, diesen Optimierungsweg weiter erfolgreich zu beschreiten.

Günter Penzenauer
R&B Consulting Group

Vorwort

Die ersten von Menschen gebauten Werkzeuge - damals, in der Steinzeit - waren nicht sonderlich kompliziert. Ausgefeilte Baupläne waren bei den glücklichen Ingenieuren dieser Zeit nicht groß in Mode: Man nahm einen Stein und hackte einfach auf einen anderen ein, bis dieser eine gefällige Form hatte.

Menschen bauen immer komplexere und größere Werkzeuge.

Später, als Baumeister und Ingenieure der Antike, der Renaissance und des Industriezeitalters große, komplizierte Gebäude und Maschinen zu errichten begannen, wurde das Zusammenwirken mehrerer Entwickler (z.B. Baumeister und seine Gesellen) und die Abstimmung mit dem Auftraggeber (z.B. König, Fürst, Fabrikant) unabdingbar. Damit wurde auch das gemeinsame Verstehen des geplanten Werkes wichtig, und so kamen ***Baupläne und Modelle*** ins Spiel.

Ausschlaggebend: Verstehen, wie das zu bauende Werk verwendet wird.

Beim Bauen von Werkzeugen und Maschinen ist das Verstehen der späteren ***Verwendung*** ausschlaggebend dafür, ob das Produkt praxistauglich wird. Das Wahrnehmen des Anwendungszwecks und der Anwendungsweise ist Vorbedingung für korrekte Entscheidungen der Konstrukteure.

Alles andere muss sich in die Weise, wie ein Produkt verwendet wird, einfügen: Die ***Funktionsweise***, die ***Form***, die ***Konstruktionsmittel***.

Dieses Buch handelt von der Zusammenarbeit und vom gegenseitigen Verstehen von Menschen, die gemeinsam komplizierte elektronische Maschinen beauftragen und bauen. Diese Maschinen, die man nicht in die Hand nehmen und anfassen kann, nennt man ***Software***.

Software ist eine Maschine, die man nur über ihre Benutzeroberfläche anfassen kann.

Software ist nicht aus Holz und nicht aus Stein. Die Gesetze der newtonschen ***Mechanik*** gelten für sie nicht; man kann sie also nicht über ihre mechanischen Eigenschaften wahrnehmen. Softwarefunktionen quietschen nicht beim Starten, Menü- und Symbolleisten biegen sich nicht unter der Last ihres Inhalts durch, und Dialogseiten können einem nicht auf den Fuß fallen.

Wenn Software solche mechanischen Eigenschaften hätte, könnten diese durchaus Aha-Erlebnisse bei Anwendern und Entwicklern auslösen.

Verwenden ist der Schlüssel zum Verstehen.

Das Zugangstor zum Wahrnehmen, Auslösen und zum Definieren der Softwarefunktionen ist jedoch allein die Art und Weise, in der man Software ***verwendet*** - ihre ***Benutzerschnittstelle*** (in Computerenglisch: ***User Interface***, abgekürzt „UI“).

Da das User Interface eine Schlüsselrolle spielt, ist demnach eine ***User Interface-orientierte Softwarearchitektur*** nötig, um die Softwareoberfläche im Entwicklungsvorgehen und in der Konstruktionsweise der Software stärker zu fokussieren, als es die heutigen Architekturen tun.

Das Buch über User Interface-orientierte Architektur

User Interface-orientierte Softwarearchitektur zeigt, wie man Softwareanwendungen für Menschen baut, indem man das Verwenden der Software in den Mittelpunkt der Abstimmung zwischen Auftraggeber und Entwicklern stellt. Die Verwendung findet über die Oberfläche statt - für den Anwender ist die Oberfläche die Software. Beschreibt man die Oberfläche im Detail - also das, wie sich die Software im Dialog mit dem Anwender verhält - dann ist auch die Funktion beschrieben, und Auftraggeber und Entwickler werden sich einig, was gebaut wird.

Bauentwurfslehre

Das Buch zeigt auf, welche Begriffe, Redewendungen und ***Beschreibungsstrukturen*** benötigt werden, um die Verwendung einer Softwareanwendung (nicht nur Use Cases, sondern die komplette Oberfläche einer Büroanwendung mit jedem Pieps, den sie von sich gibt) eindeutig und missverständnisfrei zu beschreiben.

Fokus auf kommerzielle Anwendungen

Der Fokus des Buchs liegt auf Oberflächen für ***kommerzielle***, ***wirtschaftlich orientierte Software***. Diese Anwendungen, etwa Verwaltungssoftware, Beratungs-, Verkaufs- und Buchungssysteme unterstützen Arbeitsabläufe der Unternehmen durch elektronische Workflows, dialoggestütztes Ausfüllen von Formularen sowie durch verschiedene Such- und Auswertungsfunktionen.

Heute werden solche Anwendungen in der Regel mit grafischen Benutzeroberflächen als Windows- oder als Browserapplikation implementiert. Diese Art von Softwareoberflächen kommt nach meiner Einschätzung in 90% aller geschäftorientierten Anwendungen vor.

Die UI-orientierte Architektur geht zum Teil auch auf Fragestellungen von sprachgesteuerten Dialogen und von eingebetteten Systemen (z.B. Radionavigationssysteme).

Andere User Interface-Arten wie freie Dialoge, Virtual Reality, Augmented Reality, Spiele, KI-Systeme sowie neue, experimentelle UI-Formen werden nur am Rande erwähnt, weil sie in der Praxis kommerzieller Projekte vergleichsweise selten vorkommen.

Softwareoberflächen gibt es heute überall. Sie werden aber mit wenig methodischer Unterstützung gebaut - gewissermaßen nach dem Freihandprinzip und nach der Art einer Hundehütte zusammengezimmert.

Verwenden ist der Schlüssel zum Beschreiben

Wie aber entwirft und konstruiert man das, was an der Oberfläche einer Software geschieht, oder geschehen soll, eindeutig und hinreichend genau? Mit welcher Kombination aus Methoden und Werkzeugen lässt sich das Geflecht aus Verwenden, Dialog, Wirkung, Erscheinen und Funktionieren planen und beschreiben?

Darüber gibt dieses Buch über User Interface-orientierte Architektur Auskunft und Anleitung. Das Buch zeigt Mittel und Wege auf, um Benutzeroberflächen strukturierter und professioneller zu entwickeln.

Mit dem Know-how aus dem Buch ist der Leser in der Lage, Anwendungen so zu entwickeln, dass jederzeit Klarheit und Sicherheit über die Funktionsweise und die Funktionalität der entwickelten Anwendung gegeben ist. Dadurch können Kommunikationsaufwände gesenkt, die Budgets und Zeitvorgaben des Entwicklungsprojekts abgesichert und die Anzahl der Änderungsaufträge minimiert werden.

Paul Chlebek
Affing, im Juni 2006

Inhaltsverzeichnis

Kapitel 1: Einführung 1

1.1 Mensch und Software 1

1.2 Zielgruppe 5

1.3 Nutzen der UI-orientierten Architektur 6

1.4 Benutzeroberfläche in den Mittelpunkt rücken 8

1.5 Beispiel für UI-orientiertes Entwickeln 11

1.6 Inhalte der User Interface-Architektur 13

1.7 Lesen von UML-Diagrammen 21

1.8 Zusammenfassung 24

Kapitel 2: Oberfläche 27

2.1 Historische Entwicklung der Softwareoberflächen 29

2.2 Oberfläche gliedern 33

2.3 User Interface-Arten 39

2.4 Funktionsweise einer Softwareoberfläche 42

2.5 Informationsgehalt des UI 50

2.6 Kontrollelemente 52

2.7 User Interface-Logik 54

2.8 Nutzen 56

Kapitel 3: Werkzeuge 57

3.1 Typische Phänomene des UI-Entwickelns 58

3.2 Anforderungen an UI-Werkzeuge 61

3.3 Bewertungsschema 63

3.4 Werkzeugarten 70

3.5 Office-Werkzeuge 70

3.6 Darstellungsorientierte Werkzeuge 71

3.7 GUI-Toolkits und GUI-Bibliotheken ... 71
3.8 Grafische Ablaufmodellierungswerkzeuge ... 72
3.9 Werkzeuge für eingebettete Systeme ... 72
3.10 Nutzen ... 81

Kapitel 4: Skizzieren ... 83
4.1 Methodenwahl ... 84
4.2 Systemgrenzen abstecken ... 89
4.3 Ablaufstruktur formen ... 92
4.4 Logische Schritte modellieren ... 95
4.5 Dialoginhalte abgrenzen ... 100
4.6 Nutzen der UI-Skizzen ... 101

Kapitel 5: Detaillieren ... 105
5.1 Perspektiven des UI-Entwickelns ... 106
5.2 Iterationsmodell ... 109
5.3 Fertigstellungsgrad messen ... 114
5.4 Nutzen der UI-Detailmodelle ... 116

Kapitel 6: Sprache ... 119
6.1 Use Cases des UI-Modellierens ... 121
6.2 Die State Charts Notation ... 124
6.3 Konzepte für eine UI-Beschreibungssprache ... 129
6.4 Datenmodell für UI-Spezifikationen ... 135
6.5 UI-Beschreibungssprache Lucia ... 138
6.6 Umfang der Lucia-Sprache ... 146
6.7 Lucia-Grammatik ... 149
6.8 Nutzen ... 162

Kapitel 7: Architektur ... 163
7.1 Integration in die Entwicklungsumgebung ... 163
7.2 Das MVC-Entwurfsmuster ... 165
7.3 Model Driven Architecture (MDA) ... 169
7.4 UI-orientierte Laufzeitkomponenten ... 170
7.5 Ausleiten von Prototypen ... 172

7.6 Weitere Transformationen ... 172
7.7 Nutzen ... 172

Kapitel 8: Funktionsmodellierung ... 175
8.1 Abhängigkeiten zwischen Bedienung und Funktion ... 176
8.2 Kollaboratives Modellieren ... 177
8.3 Oberflächeneigenschaften mit UML-Modellen absichern ... 179
8.4 Funktions- und Datenmodelle ableiten ... 180
8.5 Nutzen ... 181

Kapitel 9: Prozesssteuerung ... 185
9.1 Prozesslogik und Situationen ... 186
9.2 Ablaufsteuerungsdaten extrahieren ... 188
9.3 Prozessbeschreibungen im UI abbilden ... 191
9.4 Ablaufstrukturen visualisieren ... 192
9.5 Nutzen ... 193

Kapitel 10: Dokumentation ... 195
10.1 Entwicklungsdokumente generieren ... 196
10.2 Benutzerdokumente generieren ... 197
10.3 UI-Änderungen nachvollziehen ... 198
10.4 Nutzen ... 198

Kapitel 11: Testen ... 201
11.1 Arten von UI-Tests ... 202
11.2 Testszenarien generieren ... 204
11.3 Navigation beim UI-Test ... 205
11.4 Nutzen ... 206

Kapitel 12: Projektmanagement ... 209
12.1 Arbeiten im Kontext der UI-Anforderungen ... 210
12.2 UI-orientiertes Vorgehen ... 211
12.3 Integration sicherstellen ... 211
12.4 Rollen und Aufgabenteilung bei der UI-Entwicklung ... 215
12.5 Nutzen ... 221

Kapitel 13: Lebenszyklusmanagement 223

13.1 Reengineering und Migration 225

13.2 UI-Beschreibung wieder verwenden 227

13.3 Zusammenfassung und Ausblick 228

Literaturverzeichnis 231

Sachwortverzeichnis 233

Abbildungsverzeichnis

Bild 1: Software macht Computer für Menschen verwendbar. 1
Bild 2: Software anhand der Verwendens der Oberfläche verstehen 2
Bild 3: Oberflächen sind Bedienpulte für Funktionen der Software 4
Bild 4: Typisches User Interface einer kommerziellen Dialoganwendung 4
Bild 5: Navigieren in kommerziellen Anwendungen 5
Bild 6: Leute, die am Entstehen einer Software beteiligt sind 6
Bild 7: Gemeinsames Entwickeln macht Pläne und Modelle notwendig 6
Bild 8: Dialogbaupläne helfen, Missverständnisse zu vermeiden. 9
Bild 9: Form and Function follows Use Case 10
Bild 10: Verwendung, Funktion, Technik, Form 11
Bild 11: Entwickler modellieren vorwiegend das Innenleben der Software 11
Bild 12: Leitfaden der User Interface-orientierten Architektur 16
Bild 13: User Interfaces machen den Großteil einer Software aus 27
Bild 14: Eine Softwareoberfläche besteht aus verschiedenen Teilen 29
Bild 15 Themengliederung für das Beschreiben einer Oberfläche 37
Bild 16: Informationsgeflecht einer UI-Beschreibung 37
Bild 17: Erwartungen des Anwenders und der Oberfläche 40
Bild 18: Empfangen, Interpretieren, Verarbeiten, Manifestieren 45
Bild 19: Software kann man nicht mechanisch wahrnehmen 47
Bild 20: Dialog zwischen Anwender und Oberfläche 48
Bild 21: Mensch-Maschine-Dialog aus der Sicht des UI 49
Bild 22: Use Cases - Anwendersicht 51
Bild 23: Use Cases - Sicht der Oberfläche 52
Bild 24: Use Cases - Sicht der funktionalen Anwendung 53
Bild 25: Ablaufnetz, Dialogseiten und Kontext 59
Bild 26: Wissen, worauf es beim UI-Werkzeug ankommt 61
Bild 27: Bewertungskriterien für eine UI-Entwicklungsumgebung 70
Bild 28: Beispiel für eine Bewertung von UI-Werkzeugen. 74
Bild 29: Durchgängige UI-Werkzeuge haben eigene Philosophien 77
Bild 30: Guide View-Editor 81
Bild 31: Guide State-Editor 81
Bild 32: IAV Teledrive - Spezifikation 84
Bild 33: IAV Teledrive - Simulation 85
Bild 34: Projektbeteiligte bringen mentale Modelle der Anwendung mit 89

Bild 35: Systemgrenzen am Beispiel einer Kundenverwaltung 95
Bild 36: IDEF0 Activity Box 96
Bild 37: Aktivitäten verketten 97
Bild 38: Dekomposition und Border Lines 97
Bild 39: Prozessgliederung, Dialogfluss und Dialogseiten 99
Bild 40: Struktur der logischen Schritte in der Kundenverwaltung 101
Bild 41: Grafische Darstellung für logische Schritte 102
Bild 42: Wichtigste Kennzeichen für logische Schritte 103
Bild 43: Aktiver Schritt und hierarchisch semi-aktive Schritte 105
Bild 44: Aktiver Schritt und orthogonal semi-aktive Schritte 106
Bild 45: Vorgehenskarte für das Skizzieren der Oberfläche 109
Bild 46: User Interfaces werden schrittweise entwickelt 111
Bild 47: Verschiedene Sichten auf die Oberfläche 113
Bild 48: UI evolutionär aus wechselnden Perspektiven entwickeln 115
Bild 49: Rational Unified Process 116
Bild 50: UI Inception, Elaboration, Construction, Transition 116
Bild 51: Iterationsmodell für die UI-Entwicklung 117
Bild 52: Leitfaden für das Detaillieren der UI-Beschreibung 123
Bild 53: Ablauf einer UI-Iteration 123
Bild 54: Einsatz einer UI-Sprache für praxisübliche Oberflächen 126
Bild 55: Syntax, Semantik, Informations-Modell 126
Bild 56: Von einer Beschreibungshilfe zum MDA-Werkzeug 127
Bild 57: Anwendungsfall-Gruppen beim UI-Modellieren 128
Bild 58: UI-Außensicht und UI-Innensicht 130
Bild 59: Zustände in einer State Chart 132
Bild 60: State Chart: Zustände mit Aktivitäten und Transitionen 133
Bild 61: Auslöser, Bedingung und Reaktion in einer Transition 133
Bild 62: Orthogonale Zustände 133
Bild 63: Flache und tiefe Historie 134
Bild 64: State Chart für einen Schalter 134
Bild 65: Kernentitäten eines Informationsmodells für User Interfaces 143
Bild 66: Das Informationsmodell kapselt Notation und Laufzeitformat 144
Bild 67: Inhalte und Verflechtungen der UI-Perspektiven 145
Bild 68: Spezialisten interessieren sich für unterschiedliche Softwareteile 147
Bild 69: Überblick über Objekttypen in Lucia 148
Bild 70: Sprachstruktur von Lucia 157

Bild 71: Fehlende Integration der UI-Modelle in der Architektur 174
Bild 72: UI-Spezifikation und UI-Codierung laufen Hand in Hand. 175
Bild 73: MVC-Entwurfsmuster 176
Bild 74: MVC und Zuordnung von Entwicklungs-Themen 178
Bild 75: Komponenten-Modell für eine UI-orientierte Architektur 182
Bild 76: Oberfläche, Funktionen und Daten sind Teile eines Ganzen 185
Bild 77: Gegenseitige Abhängigkeiten von Bedienung und Funktionen 187
Bild 78: Klassengerüst für die Kundenverwaltung 191
Bild 79: Vorgehenskarte für synchrone UI- und UML-Modellierung 192
Bild 80: Abläufe und Regeln sind implizit im Programmcode eingebaut 195
Bild 81: Aus einem UI-Modell kann man Dokumente ableiten. 206
Bild 82: Verwerten des UI-Modells zum Erstellen von Dokumenten 209
Bild 83: Mit UI-Modellen sind einfache UI-Tests automatisierbar. 211
Bild 84: Klassifikation von automatisierten Tests 212
Bild 85: Vorgehenskarte für das automatisierte Testen des UI 214
Bild 86: Artefaktekette beim UI-Test 217
Bild 87: Beim UI fühlen sich viele kompetent. 219
Bild 88: User Interface-Entwicklung ist ein interdisziplinäres Thema 220
Bild 89: Funktions- und Daten-Entwickler grenzen das User Interface gerne aus. 222
Bild 90: UI-bezogene Rollen und Aufgaben im Entwicklungsprozess 226
Bild 91: Use Cases Map der UI-Entwicklung - Bedienkonzept 227
Bild 92: Use Cases Map der UI-Entwicklung - Erscheinungsbild 227
Bild 93: Use Cases Map der UI-Entwicklung - Inhalte 228
Bild 94: Use Cases Map der UI-Entwicklung - Methoden 228
Bild 95: Use Cases Map der UI-Entwicklung - Umsetzen 229
Bild 96: Use Cases Map der UI-Entwicklung - Testen 229
Bild 97: Tool Chain für die UI-Entwicklung 231
Bild 98: Gewachsene Anwendungen werden immer weniger überschaubar 233
Bild 99: Neue Software kann man unterschiedlich empfinden. 234
Bild 100: Etablierte Softwarelösungen sind oft organisch gewachsen 235
Bild 101: UI-orientierte Architektur unterstützt Knowledge Management 240
Bild 102: Möglichkeiten der Verwendung von UI-Modellen 240

Kapitel 1: Einführung

… in dem Sie erfahren, warum Sie dieses Buch lesen sollten und was Sie in seinen Kapiteln finden.

Kapitelziele

- *Verstehen der Problemstellungen und der Lösungsansätze der User Interface-orientierten Architektur.*
- *Überblick über Themen, Zielgruppe und Aufbau dieses Buchs.*

1.1 Mensch und Software

Anwendungsprogramme sind Dinge, die Computer für Menschen nutzbar machen. Durch sie kann der Computer dem Anwender bei seinen Aufgaben helfen, z.B. Buchhaltung machen, Termine organisieren, Kundenkontakte verwalten, Grafiken und Texte erstellen.

Was wir als Anwender von einer Anwendungssoftware wahrnehmen können, ist ausschließlich ihre Benutzerschnittstelle: Die so genannte ***Softwareoberfläche*** oder das ***User Interface***.

Für Anwender ist die Oberfläche die Software

Wenn wir über eine Software sprechen, sprechen wir darüber, was wir wahrnehmen (sehen, hören, etc.) und tun können (Knöpfe drücken, Menüpunkte wählen, Daten eingeben, etc.), also eben über die Oberfläche der Software. Für Anwender ist Software nur das, was sie wahrnehmen und steuern können.

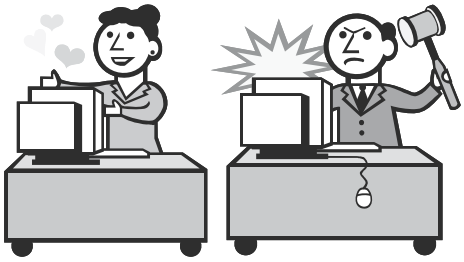


Bild 1: Software macht Computer für Menschen verwendbar.

Software soll die vielfältigen Funktionen des Computers für Menschen erschließen und leichter verwendbar machen. Durch ihre Oberfläche wird Software anfassbar und begreifbar. Hingegen ist für die Nutzer eines Computerprogramms - oder eines anderen vielseitigen Werkzeugs - dessen innerer Aufbau meist nur am Rande wichtig. Sie interessieren sich hauptsächlich dafür, ob das

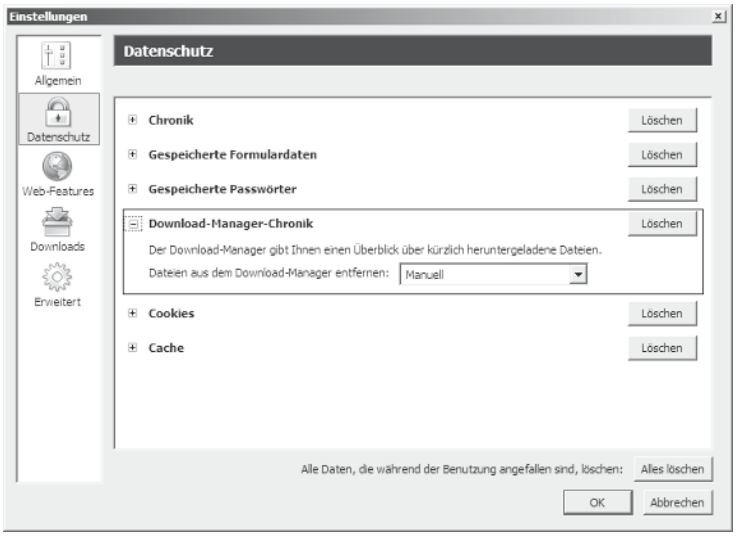
Ding gut in der Hand liegt und seinen Zweck gut erfüllt. Bei Software interessiert den Anwender die Oberfläche, bei Fahrzeugen das Fahrerlebnis, bei Maschinen das Bedienpult.

Gemeint ist: Das Verstehen einer Maschine erfolgt anhand dessen, wie die Maschine vom Anwender beim Benutzen erlebt wird. Das gilt sowohl für reale Maschinen wie einen Korkenzieher oder einen Toaster, als auch und insbesondere für virtuelle Softwaremaschinen und ihre Benutzeroberflächen, wie zum Beispiel ein Textverarbeitungsprogramm, eine Suchmaschine im Internet oder das Navigationssystem in einem Fahrzeug.

Funktionale Beschreibungen ergänzen Beschreibungen der Verwendung.

Anwendungssoftware ist ein Gerät, das nur durch das ***Verwenden*** der Softwareoberfläche für den Anwender materiell existiert. Also kann man Software hinreichend nur anhand der ***Interaktion*** des User Interface mit dem Anwender, nicht aber über den inneren Aufbau der Softwarekomponenten allgemein definieren, beschreiben, vermitteln und verstehen.

Bild 2: Software anhand der Verwendens der Oberfläche verstehen



Beschreibung des User Interface hat Vorrang vor funktionalen Spezifikationen.

In der Konsequenz heißt das aber: Datenmodelle und Flussdiagramme, Klassen- und Komponentendiagramme können eine Verwendungsbeschreibung sinnvoll ergänzen, sie können aber nicht von der Verwendungsbeschreibung losgelöst erstellt oder verstanden werden.

Daraus folgt, dass funktionale Spezifikationen und Modelle nachrangig zur Spezifikation des User Interface einer Softwareanwendung sind.

Beispiel für User Interface-Orientierung

Zum Beispiel ein Fernseher

Platt gesagt: Man kann zum Beispiel einen Fernseher nicht anhand seines Bauplans erklären, sondern nur darüber, dass man das Fernsehen als Tätigkeit beschreibt. Im ausgeschalteten Zustand sieht ein Computermonitor einem Fernsehgerät ja nicht unähnlich.

Die Aufgabe, den Fernseher anhand des Fernsehens zu beschreiben, ist ein Beispiel für das Wahrnehmen und Verstehen anhand der Verwendungsszenarien. Wir wissen, dass wir uns verschiedene Programme ansehen können, je nachdem, welche Tasten wir auf der Fernbedienung drücken. Wir wissen auch, wann was im Fernseher kommt, sofern wir eine Fernsehzeitung lesen. Wir können einen Kanal auswählen und gucken. Wenn das Antennenkabel eingesteckt ist, aha.

Am Anfang steht der Use Case

Man kann die Tätigkeit des Fernsehens nicht vermitteln, indem man den Schaltplan des Fernsehgeräts erklärt oder beschreibt, wie ein Fernsehstudio nebst Sender funktioniert. Vielmehr muss man die Interaktionen an der ***Bedienerschnittstelle*** des gedachten Anwendungssystems „Fernsehen“ aufzeigen.

Wissen über das Verwenden bestimmt die mögliche Qualität der technischen Umsetzung

Um das Fernsehen zu definieren, muss man also beschreiben, wie man fernsieht. Aus der Beschreibung der Tätigkeit und wie dabei Werkzeuge verwendet werden, leiten sich die Existenz, Funktion und Konstruktion dieser Werkzeuge ab. Am Ende dieses Ableitungsprozesses hat man den Schaltplan des Fernsehgeräts und die Fernbedienung gleich dazu.

Das Wissen darüber, wie man fernsieht, bestimmt, wie ein guter Fernseher und ein guter Fernsehsessel gemacht werden. Die Klarheit darüber, wie die Software an der Oberfläche verwendet wird, schafft Klarheit darüber, was technisch programmiert werden muss. Es funktioniert nicht umgekehrt, auch nicht bei Software.

Heute sind Softwareoberflächen des täglichen Gebrauchs ***Bedienpulte*** für die in der Software enthaltenen Anwendungsfunktionen.

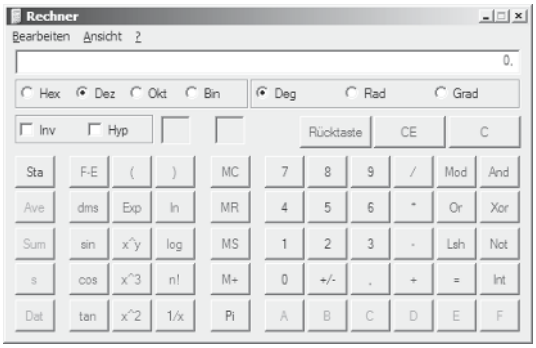

Rechner
Bearbeiten Ansicht ?
0.
Hex Dez Okt Bin
Deg Rad Grad
Inv Hyp
Rücktaste CE C
Sta F-E ( ) MC 7 8 9 / Mod And
Ave dms Exp ln MR 4 5 6 * Or Xor
Sum sin x^y log MS 1 2 3 - Lsh Not
s cos x^3 n! M+ 0 +/- . + = Int
Dat tan x^2 1/x Pi A B C D E F

Bild 3: Oberflächen sind Bedienpulte für Funktionen der Software

In den meisten Anwendungen des Büroalltags wird der Anwender durch einen Dialogfluss aus auszufüllenden Formularen geführt. Er führt dabei die von der Anwendung vorgesehenen Schritte auf verschiedenen Bildschirmmasken durch, wählt Funktionsbereiche aus bzw. steuert den Anwendungsablauf innerhalb des von der Software vorgegebenen Angebots..

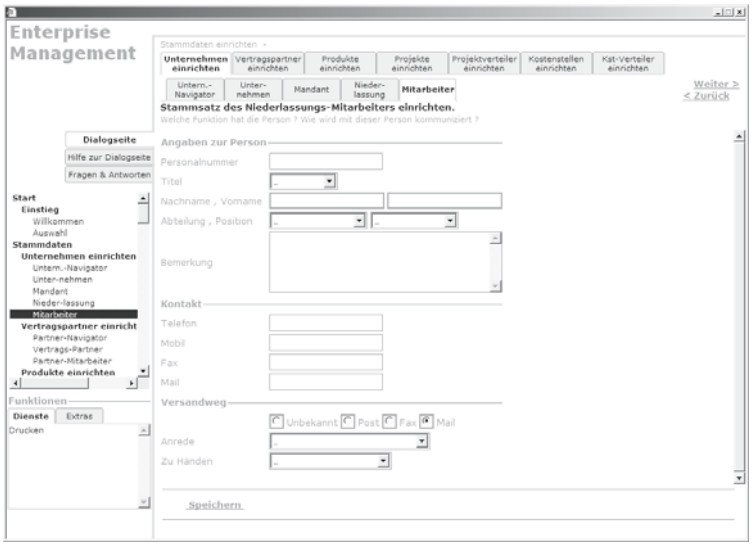


Bild 4: Typisches User Interface einer kommerziellen Dialoganwendung

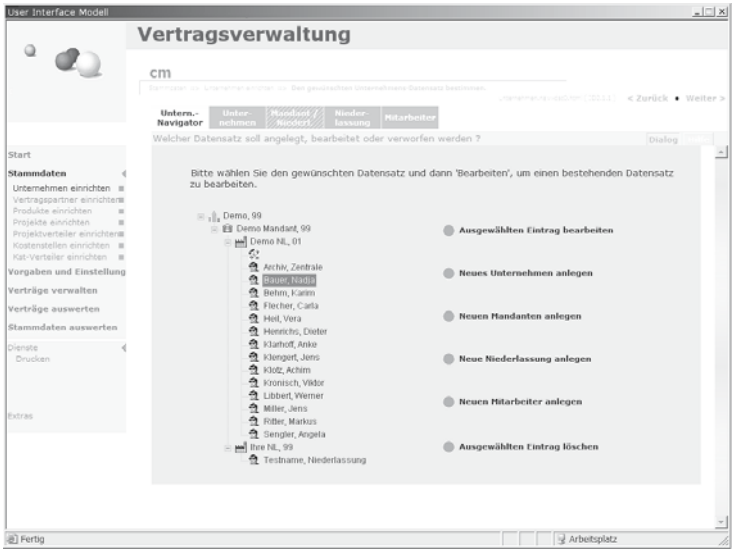


Bild 5: Navigieren in kommerziellen Anwendungen

Andere Formen von User Interfaces, z.B. der freie Dialog oder Augmented Reality sind in der Anwendungssoftware des täglichen Gebrauchs eher selten.

Eigenschaften, die über die Funktion eines guten Bedienpults hinausgehen, zum Beispiel die oft beschworene Mensch-Maschine-Kommunikation (wie zum Beispiel die zwischen Dr. David Bowman und dem Computer HAL) [Ebeling 1988, S.96f], werden von einer Anwendungsoberfläche gar nicht gewünscht.

Oder wollen Sie mit dem Navigationssystem in Ihrem Auto ausdiskutieren, ob es aus seiner Sicht Sinn macht, von A nach B über C zu fahren? Zumal eine intelligente Anwendung auch lügen darf, um den Turing Test zu bestehen [Rose 1985, S.62].

1.2 Zielgruppe

Wo Software entsteht, gibt es stets jene, die sie beauftragen und sie anschließend nutzen: ***Die Auftraggeber***. Und natürlich gibt es auch jene, die die Software herstellen: ***Die Entwickler***. Das Buch, das Sie aufgeschlagen haben, ist diesen beiden Zielgruppen zu gleichen Teilen gewidmet.

Für die, die Software herstellen, und für die, die Software beauftragen

User Interface-orientierte Softwarearchitektur ist für Leute, die Software herstellen: Programmierer, Architekten, Designer und Analytiker. Und sie ist für Leute, die die Herstellung von Software beauftragen und steuern: Projektleiter, Anwendervertreter, Anforderungssteller und Qualitätssicherer.

Zur primären Zielgruppe der User Interface-orientierten Architektur gehören:

- ***Softwareentwickler***
- ***Softwarearchitekten***
- ***Requirement Engineers (z.B. Spezifizierer und Analytiker)***

Weitere Zielgruppen des Buchs sind:

- ***Projektleiter***
- ***Anforderungssteller (z.B. Anwendervertreter und Fachexperten)***
- ***Qualitätssicherer und -prüfer***
- ***Designer***

Bild 6: Leute, die am Entstehen einer Software beteiligt sind

Es ist genau genommen ein Buch, das eine Brücke zwischen diesen Gruppen und Rollen baut. Diese Brücke sorgt für ***Verständigung*** und friedliches Zusammenleben dieser seit je her zu Missverständnissen im Umgang miteinander neigenden Völker.

User Interface-orientierte Softwarearchitektur zeigt die ***Kommunikationslücken*** zwischen den am Bau einer Softwareanwendung beteiligten Rollen auf. Zum Beispiel werden typische Verständigungsprobleme zwischen Softwarehersteller und Auftraggeber sowie Lösungsmöglichkeiten dafür aufgezeigt.

Bild 7: Gemeinsames Entwickeln macht Pläne und Modelle notwendig

Das Buch zeigt, wie die beteiligten Personen in ihren jeweiligen Rollen mit weniger Missverständnissen und Informationsverlusten gemeinsam an der Planung und Realisierung einer Software zusammenarbeiten können. Es stellt allgemeinverständliche und effiziente Möglichkeiten zur eindeutigen Beschreibung der Funktionen der Anwendung aus Anwendersicht zur Verfügung und zeigt, wie auf dieser Basis die Effizienz der Softwareentwicklung in allen Projektphasen entscheidend verbessert wird.

1.3 Nutzen der UI-orientierten Architektur

Methodensammlung für die Projektpraxis

Die User Interface-orientierte Softwarearchitektur ist auf die Praxis der Softwareentwicklung gerichtet. Das Buch aus Erfahrungen entstanden, die im Projektalltag gemacht wurden.

Kommerzielle Softwareprojekte haben stets enge Zeitvorgaben, begrenzte Budgets und müssen im vorgegebenen Umfeld mit Menschen gemacht werden, die unterschiedliche, ebenfalls begrenzte Fähigkeiten mitbringen. Man kann es sich nicht aussuchen: Auf Auftraggeberseite gibt der Marktdruck („Time To Market“) das Tempo vor; auf Herstellerseite ist das eigene technische Know-how bestimmend für die Wahl der Werkzeuge und Methoden. Auf beiden Seiten spielen die Kosten und die Verfügbarkeit von Ressourcen eine entscheidende Rolle.

Eine keimfreie Atmosphäre, in der man Lehrbuchwissen unter Laborbedingungen anwenden könnte, findet man im Projektge-

schäft nicht. Das Buch konzentriert sich daher auf alltägliche Projektsituationen, auf Beispiele aus der Praxis und auf konkrete Herausforderungen der Entwicklung von komplexen und dialogintensiven Anwendungen unter Zeit- und Erfolgsdruck. Theorien und Betrachtungen über den Besenstiel, z.B. Vorgehensmodelle und Metamodelle formaler Sprachen bleiben weitgehend außen vor.

Im Projekt zählt die Praxistauglichkeit in einem Umfeld, bei dem das Vorgehensmodell und die Werkzeuge, die Programmiersprachen und die Laufzeitplattform weitgehend vorgegeben sind. Es geht also darum, wie man mit gegebenen Mitteln und in einem gegebenen Projektumfeld das Projektziel sicherer erreicht. Best Practice eben.

Ich verspreche Ihnen nicht, dass Sie nach dem Lesen der User Interface-orientierten Architektur mehr Qualität in kürzerer Zeit erreichen oder die Projektkosten senken. Mehr und bessere Turnierkrokodile in kürzerer Zeit mit weniger Geld: Das funktioniert nur in der New Economy und die hat ja nur für wenige funktioniert.

Planungssicherheit

Ich verspreche Ihnen stattdessen, dass Sie besser schlafen können, weil Sie genauer wissen, was Sie eigentlich bauen oder gebaut kriegen. Sie schlafen umso besser, weil Sie auch sicher sein können, dass Ihr Projektpartner - Ihr Auftraggeber, Ihr Softwarehersteller - das gleiche Verständnis davon hat, wie die Software, die Sie gemeinsam erarbeiten, aussieht und funktioniert.

Sicherung der Projektziele

Die Sicherung der Projektziele wird erreicht, indem Risiken systematisch eliminiert werden. Das kann mit folgenden Maßnahmen erreicht werden.

- Abgestimmte Schnittstellen
- Erlebbarer User Interface-Entwurf
- Durchgängige und eindeutige Spezifikation von der Softwareoberfläche bis zu den Systemfunktionen.

In Bezug auf die Oberfläche kann die Sicherung der Projektziele durch folgende Eigenschaften der Oberflächenspezifikation erreicht werden.

- Formulierbarkeit
- Erlebbarkeit
- Nachvollziehbarkeit
- Durchgängigkeit

Doppelarbeit vermeiden

Sie können mit der User Interface-orientierten Softwarearchitektur keine Arbeit sparen, dafür aber Zeit.

Wenn Sie etwas bauen, z.B. einen Schreibtisch, können Sie nicht, sagen wir mal, die Tischplatte ungehobelt lassen, wenn Sie diese am Ende schleifen und polieren wollen.

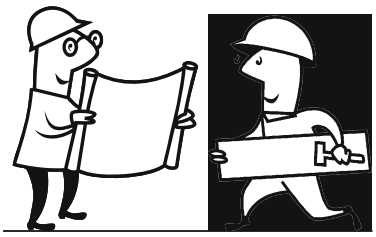
Das heißt: Alle zu einem Entwicklungsprozess gehörenden Arbeitsgänge müssen durchgeführt werden. Das Buch hilft Ihnen allerdings dabei, die Entwicklungsarbeit im gemeinsamen Kontext und mit der größtmöglichen Transparenz für alle Projektbeteiligten zu erledigen. Sie schlafen ziemlich ruhig, wenn das sichergestellt ist.

1.4 Benutzeroberfläche in den Mittelpunkt rücken

Missverständnisse bei Auftragsvergabe und Systementwurf verursachen oft erhebliche Folgekosten. Die Vermeidung von Missverständnissen lässt sich durch eindeutige Oberflächenbeschreibungen, so genannte ***Dialogbaupläne*** erreichen. Eindeutige Dialogbaupläne braucht man, um Softwareoberflächen gemäß Kundenbestellung zu programmieren.

Dialogbaupläne beschreiben das User Interface eindeutig und helfen, Missverständnisse zu vermeiden.

Bild 8: Dialogbaupläne helfen, Missverständnisse zu vermeiden.



Das Buch ist eine Anleitung zum Erstellen von Dialogbauplänen.

Die Oberfläche ist das, was Kunden und Anwender von der Software sehen; für sie ist ***das*** die eigentliche Software. Sie teilen den Entwicklern ihre Anforderungen an die Software über ihre Vorstellung von dem Geschehen an der Bedienerschnittstelle mit. Deshalb ist ein für Kunde und Softwarehersteller gleichermaßen verständlicher, eindeutiger, genauer und einvernehmlicher Bauplan der Oberfläche ausschlaggebend für den Erfolg eines Softwareprojektes.

Pläne und technische Zeichnungen sind schon beim Bau von Häusern, Schiffen, Automobilen und anderen Maschinen unabdingbar. Während der Planungsphase werden Draufsichten, Seitenansichten, perspektivische Ansichten und verschiedene Querschnitte des Bauvorhabens erstellt. Ohne sie wäre die Beauftragung, Genehmigung und Durchführung des Bauvorhabens undenkbar.

Die Nutzung steht im Vordergrund der Modellierung von Bauwerken und Maschinen

Gerade da, wo es um die Nutzung des Bauvorhabens, z.B. die Inneneinrichtung der Büro- und Wohnräume, das Cockpit des Fahrzeugs oder das Bedienpult der Maschine geht, sind zur Abstimmung der Funktions- und Verwendungsweise Modelle und Simulationen üblich.

Bei Software steht heute noch der innere Aufbau im Vordergrund

Während im Häuser-, Maschinen- und Fahrzeugbau die Sicht des Nutzers meist im Mittelpunkt steht, konzentriert sich z.B. die UML beim Softwarebau auf Querschnittsmodelle über Funktionen, Datenhaushalt und ***innere*** Abläufe. Nicht selten wird die Oberfläche nur mit Hausmitteln wie Powerpoint, Excel, State Charts und Designprototypen umrissen und die Ausgestaltung der Details dem Entwickler überantwortet. Manchmal begegnet man noch extremeren Beispielen: Die Funktionalität wird ohne Oberfläche als „Framework" entwickelt und die Oberfläche in einem Einführungsprojekt beim Kunden „drübergelegt".

Die Bedienung der Software wird damit einhergehend nicht als Mittelpunkt des Projekts, sondern eher als Aufsatz auf die technische Umsetzung der Funktionalität wahrgenommen. In der Praxis hat das allerdings zur Folge, das erst beim Entwickeln der Oberfläche die Lücken und Brüche in einem vermeintlich einsatzfähigen Framework zum Vorschein kommen.

Form follows Use Case

Der Henkel einer Tasse ist ihre Benutzerschnittstelle. Ein auf der Innenseite der Tasse angebrachter Henkel funktioniert zwar mechanisch genauso, stellt jedoch den Anwender beim Verwenden des Henkels, insbesondere in Kombination mit Heißgetränken, vor große Herausforderungen.

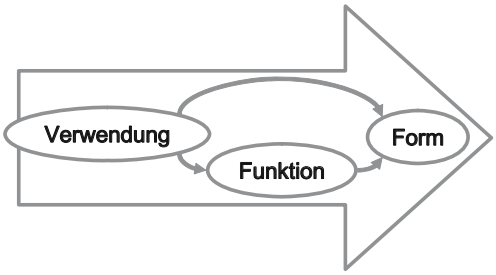
Sie kennen sicher aus eigener Erfahrung die eine oder andere Softwareanwendung, bei der Ihnen der Gedanke an eine Tasse mit Henkel innen einfallen mag.

Form follows function. Function follows Use Case

Der bekannte Satz „***Form follows Function***" kann aus meiner Sicht zu dem folgereichen Missverständnis führen, dass eine funktionale Spezifikation ein Werkzeug hinreichend beschreiben kann. Zum Beispiel soll ein Fotoapparat photographieren, aber diese Funktion wird unter Wasser anders ausfallen als auf einem Kindergeburtstag.

Es gilt daher auch: „***Function follows Use Case***". Die Form orientiert sich an der Funktion, aber auch an der Verwendung des Werkzeugs, weil die Funktion davon abhängt, wie, von wem und unter welchen Umständen das Werkzeug verwendet wird.

Bild 9: Form and Function follows Use Case



Form follows Malfunction

Man könnte auch überspitzt formulieren „***Form follows Malfunction***“, denn oft gehört es zu den Aufgaben einer Oberfläche, das Risiko von Fehlbedienung und Fehlfunktion zu senken. Damit hat eine Oberfläche neben dem Bereitstellen von benutzbaren Funktionen auch eine anleitende und eine dokumentierende Aufgabe.

Der Kernsatz „***Form and Function follows Use Case***“ eröffnet den Zugang zur folgenden Sicht auf die Zusammenhänge zwischen dem Anwenden einer Software, der Funktionalität, der technischen Umsetzung und der Benutzeroberfläche.

Es gibt verschiedene Blickwinkel des Planens: Verwendungsweise, Funktionsumfang, technische Umsetzung und Form.

Bild 10: Verwendung, Funktion, Technik, Form

þ	Verwendung	Use Case (wozu und wie wird das Softwarewerkzeug benutzt?)
@	Funktion	Inhalt (was kann die Software?)
#	Technik	Umsetzung (womit, woraus und wie wird die Software gebaut?)
Ø	Form	Oberfläche (womit wird was wie an der Software bedient?)

- þ: Das Verwenden ist vorrangig.
- @: Die Funktionen ergeben sich aus Verwendungsfällen.
- #: Die technische Umsetzung der Funktionen ergibt sich aus der verfügbaren Technologie.
- Ø: Die Oberfläche ergibt sich aus den Verwendungsfällen, wird aber von der technischen Umsetzung beschränkt.

Üblich sind heute Modelle der inneren Struktur der Software. Modelle der Außenwirkung der Software kommen meist nur in Form von UI-Prototypen vor.

Bild 11: Entwickler modellieren vorwiegend das Innenleben der Software

Verwunderlich ist: User Interfaces stehen im Mittelpunkt des Interesses der Auftraggeber und der Anwender, doch von den Entwicklern werden sie nicht selten wie Schmuck am Nachthemd behandelt. Brauchen wir nicht dringend User Interface-Modelle und Dialogbaupläne im Zentrum der Aufmerksamkeit der Softwareauftraggeber und der Softwarehersteller? Wo doch der Erfolg der Anwendung allein von ihrer Verwendbarkeit abhängt: Die Erfahrungen aus Projekten, in denen die Oberflächenentwicklung zu kritischen Verzögerungen geführt hat, sprechen eindeutig dafür.

Dialogbaupläne sollten zudem ***mehr*** als Freihandzeichnungen sein: Klar festgelegte Modelle, die in Bezug auf die Oberfläche der Software so exakt und eindeutig wie technische Zeichnungen oder CAD-Modelle von Maschinen oder Gebäuden sind.

1.5 Beispiel für UI-orientiertes Entwickeln

Haben Sie beruflich mit dem Erstellen von Softwareanwendungen zu tun? Das folgende Beispiel wird Ihnen zeigen, was User Interface-orientierte Architektur bringt und was Entwickler, Planer und Auftraggeber mit ihr erreichen können.

Szenario 1 stellt ein implementierungsorientiertes Vorgehen vor.

Szenario 2 stellt ein User Interface-orientiertes Vorgehen vor.

Szenario 1: BetaSoft & BetaRent

Beispiel für implementierungsorientiertes Vorgehen

Die Firma BetaSoft erhält von der Autovermietung BetaRent den Auftrag, eine Kundenverwaltung zu erstellen.

Nachdem der Auftragsinhalt vom Hersteller analysiert wurde, wird gemeinsam mit dem Kunden die ***Realisierungsplattform*** abgestimmt. Dabei wird schnell ersichtlich, dass die Kundenverwaltung auf der gewünschten Realisierungsplattform nicht so, wie vom Kunden gewünscht, umgesetzt werden kann. Sodann werden die Anwendungsszenarien unter den gegebenen technischen Rahmenbedingungen konkretisiert.

Bei der ***Detaillierung der Anforderungen*** werden zuerst die gewünschten Prozesse und dann die einzelnen Funktionen aufgenommen. In mehreren Iterationen werden dann die Anwendungsfälle und die Ablaufvarianten beschrieben. Ebenso wird beschrieben, wie die Anwendung in einzelnen Ausnahmefällen reagiert. Das Klassenmodell, die Ablaufdiagramme und die Fallunterscheidungen werden kontinuierlich mit BetaRent abgestimmt, bis das Konzept steht.

Erst dann wird implementiert.

Das Vorgehen ist an der technischen Funktion der Software orientiert:

- 1: Zielsetzung und Randbedingungen ermitteln
- 2: Use Cases im technischen Kontext ermitteln
- 3: Hauptabläufe bestimmen
- 4: Iterative Detaillierung der Use Cases und der Abläufe; Ausnahmebehandlung
- 5a: Fertigstellen und Abnahme des Fachkonzepts
- 5b: Entwürfe für Bildschirmmasken fertig stellen
- 6: Implementierung und Test der Software
- 7: Nachbesserungen und Change Requests

Das Projekt verläuft angespannt.

Der Kunde ist über den Entwicklungsstand unsicher und fühlt sich außer Stand, anhand der Konzeptpapiere zu prüfen, ob die Lieferung der Beauftragung entspricht. Beim Test wird festgestellt, dass nicht eindeutig feststeht, ob die gelieferte Anwendungsoberfläche die im Konzept formulierten Anforderungen erfüllt. Es werden verschiedene Änderungsanforderungen und Aufträge formuliert, die die Anforderungen an die Bedienung und die Oberfläche präzisieren.

Das Projekt überschreitet den Zeitrahmen und das Budget erheblich.

Szenario 2: AlphaSoft & AlphaRent

Beispiel für UI-orientiertes Vorgehen

Die Firma AlphaSoft erhält von der Autovermietung AlphaRent den Auftrag, eine Kundenverwaltung zu erstellen.

In Interviews werden zunächst der Einsatzzweck und die ***Einsatzszenarien*** (Use Cases) abgesteckt. Dabei wird schnell ersichtlich, dass die Kundenverwaltung Baustein einer größeren Lösung mit Bestellsystem und Fuhrparkverwaltung sein soll, die der Auftraggeber stufenweise zu einem Gesamtsystem ausbauen möchte.

AlphaSoft steckt daher gemeinsam mit dem Auftraggeber AlphaRent die ***Grenzen*** des von AlphaRent gewünschten Gesamtsystems ab und nimmt die grundsätzlichen Anforderungen an die Gesamtfunktionalität auf. So wird sichergestellt, dass die Entwickler der Kundenverwaltung den zukünftigen Gesamtkontext kennen und berücksichtigen können. Sodann werden die Anwendungsszenarien der Kundenverwaltung verfeinert.

Bei der ***Detaillierung der Anforderungen*** werden zuerst die grobe Abfolge der Dialogmasken und die Maskeninhalte aufgenommen. In mehreren Iterationen werden dann die Dialogelemente und die Anordnung der Informationen auf den Masken beschrieben. Ebenso wird bei der weiteren Detaillierung beschrieben, wie die Anwendung beim Bedienen der einzelnen Dialogelemente reagiert. Die Masken, Abläufe und Interaktionen werden kontinuierlich mit AlphaRent abgestimmt, bis die Anwendungsoberfläche steht. Begleitend werden Klassendiagramme, Aktivitätsdiagramme und Sequenzdiagramme modelliert.

Erst dann wird implementiert.

Das Vorgehen ist an der fachlichen Verwendung der Software orientiert:

- 1: Systemgrenzen abstecken
- 2: Dialogabfolge im Kontext der Verwendung ermitteln
- 3: Schlüsselfelder der Masken bestimmen
- 4: Iterative Detaillierung des UI
- 5a: Fertigstellen und Test der UI-Bedienung
- 5b: Funktionsmodelle fertig stellen
- 6: Implementierung und Test der Funktionen
- 7: Komfortverbesserungen und Erweiterungen

Das Projekt verläuft erfolgreich und in konstruktiver Atmosphäre.

Der Kunde hat jederzeit Kenntnis des Entwicklungsstands und kann die Lieferungen gegen die vereinbarte Oberfläche prüfen. Beim Test werden die Abweichungen von der gemeinsam festgelegten Bedienung festgestellt und behoben.

Das Projekt wird in Time und in Budget fertig gestellt.

1.6 Inhalte der User Interface-Architektur

Die User Interface-orientierte Architektur ist eine spannende Geschichte über die ***Zusammenarbeit von Menschen*** bei der Entwicklung von Softwareanwendungen. Sie berichtet über die unterschiedlichen Sichten der beteiligten Personen und zeigt, wie man Kommunikationsbrüche im Entwicklungsprojekt überwindet, um gemeinsam zu gewinnen.

Das Projektgeschäft ist voller Untiefen und Fußangeln. Enge Zeitpläne, fachliche und technische Restriktionen, wechselnde Anforderungen und verdeckte Ziele der Beteiligten sind nur einige Beispiele für die Gefahren, denen ein Projekt laufend ausgesetzt ist. Mit dem Buch haben Sie einen Kompass für die benutzerzentrierte Entwicklung dialogintensiver Anwendungen in der Hand. Es wird Sie in allen Projektphasen unterstützen und helfen, die Risiken mit klaren Strukturen und rollenübergreifendem Wissensmanagement zu überbrücken.

Leitfaden

Das Buch ist nach Projektphasen gegliedert

Die Kapitel der User Interface-orientierten Architektur sind nach den Durchführungsphasen eines Entwicklungsprojekts gegliedert. Jede Projektphase hat einen anderen Schwerpunkt, so auch die Kapitel des Buchs, das Sie in der Hand halten. Die Kapitel bauen nicht zwingend aufeinander auf. Ich empfehle Ihnen aber, es der Reihe nach zu lesen. So erschließen sich das Gesamtbild und die Zusammenhänge leichter. Später können Sie die einzelnen Kapitel, vor allem die Definitionen der UI-Beschreibungssprache, als Nachschlagewerk verwenden.

Die Buchkapitel beleuchten die Phasen eines Softwareprojektes und zeigen auf, wie die User Interface-orientierte Architektur in jeder Phase konkret hilft. Durch die Verknüpfung der Informationen und durch Wissensmanagement in einer durchgängigen Informationsstruktur entsteht aber auch ein phasenübergreifender Nutzen.

Innerhalb der einzelnen Kapitel werden die vorgestellten Themen anhand von Beispielen und Vergleichen aus kommerziellen Softwareprojekten verdeutlicht. Am Ende jedes Kapitels finden Sie im Abschnitt „Nutzen“ eine Zusammenfassung der vorgestellten Inhalte und konkrete Handlungsempfehlungen für die Praxis.

Die Kapitel „Einführung“, „Oberfläche“ und „Werkzeuge“ geben einen Überblick über das Entwickeln von User Interfaces.

Die Kapitel „Skizzieren“, „Detaillieren“, „Sprache“, „Architektur“, „Funktionsmodellierung“, „Prozesssteuerung“ und „Testen“ gehen auf verschiedene Projektphasen und spezielle Entwicklungsthemen ein.

Die Kapitel „Projektmanagement“ und „Lebenszyklusmanagement“ beleuchten das Planen, Organisieren und Koordinieren von Tätigkeiten in UI-orientierten Projekten.

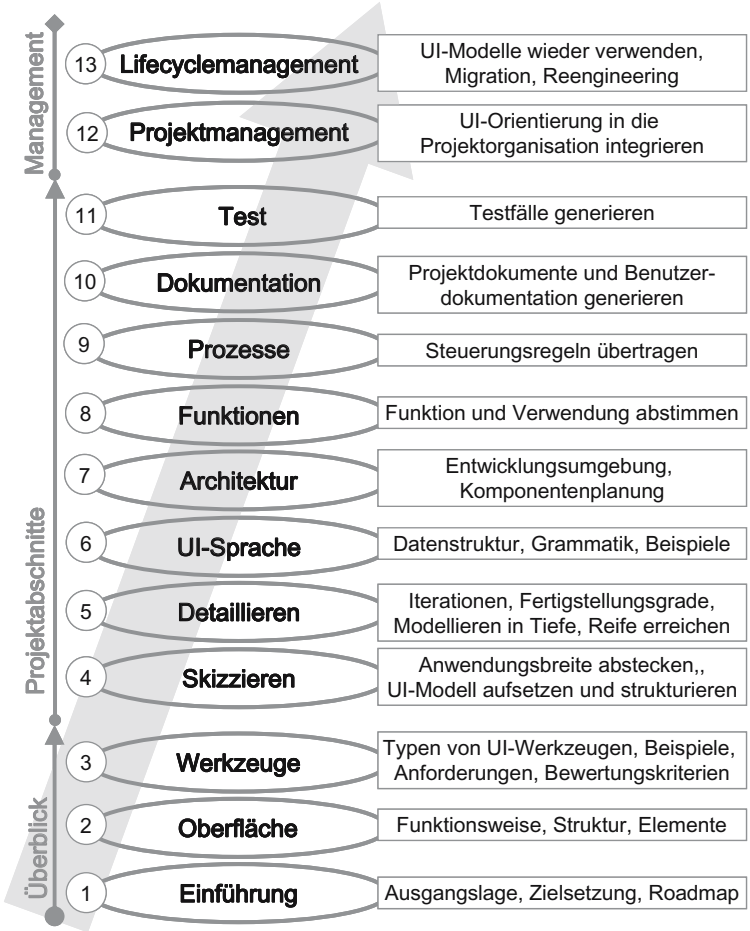


Bild 12: Leitfaden der User Interface-orientierten Architektur

Der Leitfaden führt kein neues Vorgehensmodell ein, sondern zeigt auf, wie man innerhalb etablierter Vorgehensmodelle einen deutlicheren Fokus auf das User Interface legt. Das Buch beschreibt Methoden der UI-Entwicklung. Aspekte wie Usability und Grafikdesign werden nur am Rande behandelt.

Kapitelübersicht

Einführung

Kapitel 1: „***Einführung***“, in dem Sie erfahren, weshalb und für wen das Buch geschrieben wurde.

Die Einführung erläutert den Bedarf für und den Zweck der User Interface-orientierten Architektur. Es wird die Zielgruppe, der inhaltliche Rahmen und die Struktur des Buchs beschrieben.

Ebenso finden Sie hier grundlegende Beispiele, welche klären, was die UI-Orientierung erreichen soll.

Nach dem Durchlesen des Einführungskapitels werden Sie einen Überblick über das Potential und die Bedeutung der User Interface-orientierten Architektur für Ihre Projekte haben.

Was genau ist ein User Interface?

Kapitel 2: „***Oberfläche***“, in dem wir uns die Benutzeroberfläche vornehmen und herausfinden, aus welchen verschiedenen Teilen sie besteht. Das Kapitel beinhaltet einen geschichtlichen Überblick über den Werdegang der User Interface-Entwicklung und eine Klassifizierung verschiedener Oberflächentypen.

Im Kapitel „Oberfläche“ wird eine Übersicht der Bestandteile einer Oberfläche zusammengestellt und diese Bestandteile zueinander in Beziehung gesetzt. Das Ergebnis dieser Begehung ist die Aufteilung der Oberfläche in mehrere logische Bereiche. Diese Aufteilung ermöglicht eine grobe Gliederung für eine Oberflächenbeschreibung, die man zum Beispiel als Checkliste oder als Strukturierungshilfe nutzen kann. Leitfragen zu einzelnen Aspekten einer Oberfläche versetzen Sie in die Lage, verschiedene Teile der Oberfläche voneinander zu unterscheiden und ihr Zusammenwirken zu verstehen.

UI-Werkzeuge auf dem Markt

Kapitel 3: „***Werkzeuge***“, in dem einige auf dem Markt erhältliche Werkzeuge zur UI-Modellierung und ihre Philosophien vorgestellt werden.

Wir verwenden die im vorhergehenden Kapitel aufgestellte Gliederung der Bestandteile einer Oberfläche, um unterschiedliche UI-Werkzeuge miteinander zu vergleichen. Hinter den Werkzeugen stehen auch verschiedene Philosophien, wie die Werkzeughersteller die Oberfläche in Bauabschnitte gliedern. Der Überblick stellt verschiedene UI-Werkzeuge vor, zeigt Unterschiede und Gemeinsamkeiten in der Herangehensweise an die Oberflächenmodellierung sowie Potentiale und Einschränkungen der hinter den Werkzeugen stehenden Konzepte.

Das Kapitel enthält auch eine Aufstellung der Anforderungen an UI-Werkzeuge sowie ein Bewertungsschema für die Bewertung von solchen Werkzeugen. Diese Hilfsmittel sind zum Beispiel bei der Auswahl von UI-Werkzeugen für den Projekteinsatz nützlich.

Verwendung der Software verstehen und UI planen

Kapitel 4: „***Skizzieren***“, das sich der Antwort auf die Frage widmet, wie man den Umfang und die Struktur der Anwendung absteckt.

- Wie man mit dem Entwurf der Anwendungsoberfläche beginnt.

- Wie man Funktionen und deren Verwendung an der Oberfläche skizziert.
- Wie man einen Überblick über die gesamte Breite der Anwendung gewinnt.
- Wie man bei den ersten Iterationen die überdetaillierte Beschreibung der Dialoge vermeidet.

Zum Skizzieren von Funktion und Oberfläche kann eine aus der UI-Gliederung entwickelte Informationsstruktur und Beschreibungsformalismen eingesetzt werden.

Am Beispiel der Kundenverwaltung werden wichtige Praxisschritte vorgestellt und geübt:

- Bestimmen der Systemgrenzen und
- Entwicklung des UI in Iterationen.

Die im Kapitel vorgestellten Vorgehensweisen sind vor allem in den frühen Phasen eines Softwareprojekts (Inception, Elaboration) nützlich. Die einfache Beschreibungsstruktur für User Interfaces hilft beim Aufsetzen einer robusten Systembeschreibung.

User Interface spezifizieren

Kapitel 5: „***Detaillieren***“, das den Weg von einem groben Anwendungsgerüst bis zu einer vollständigen Spezifikation der Inhalte und des Verhaltens der Anwendung aufzeigt.

- Wie man in der Breite und in der Tiefe modelliert und man eine gleichmäßige Tiefe behält.
- Wie man bewertet, wie weit der UI-Entwurf fortgeschritten ist.
- Wie man die Implementierungsreife der UI-Beschreibung erreicht und erkennt.

Der Leitfaden für die stufenweise Detaillierung der UI-Beschreibung unterstützt Sie beim iterativen Vorgehen. Klare Kriterien für die Inhalte der Iterationen ermöglichen das Erreichen und Behalten des gleichen Detaillierungsgrads in der gesamten Anwendungsbeschreibung.

Vom Modellieren in der Breite zum Modellieren in der Tiefe

Beim Skizzieren der Funktionen und der Oberfläche wird in der Breite modelliert, das heißt flach, aber mit dem Ziel einer vollständigen Übersicht.

Beim Detaillieren der UI-Beschreibung erfolgt stufenweise der Übergang von der Breitenmodellierung zur Modellierung in der Tiefe. Das heißt: Aus dem groben Modell werden die Detaileigenschaften weitergeformt. Die in diesem Kapitel vorgestellten Hilfsmittel sind in der Entwurfsphase und bei der Vorbereitung der Implementierung (Elaboration, Construction) von Nutzen.

Eindeutige UI-Modelle erstellen

Kapitel 6: „***Sprache***“, in dem enthüllt wird, wie und womit man ein User Interface eindeutig beschreiben kann.

In diesem Kapitel stelle ich eine detaillierte Informationsstruktur und eine Sprache für den Bau von Benutzeroberflächen auf. Diese Struktur ermöglicht die geordnete und zusammenhängende Verwaltung aller Informationen, die in die Implementierung der Oberfläche einfließen. Zum Füllen dieser Datenstruktur wird die UI-Beschreibungssprache „Lucia“ bereitgestellt. Die Formulierungsmöglichkeiten dieser Sprache werden vorgestellt und anhand von Beispielen erläutert. Die Spezifikationssprache ermöglicht eindeutige UI-Spezifikationen und damit eine gemeinsame Abstimmungsplattform für die am Bau des User Interface beteiligten Rollen.

Das Bespiel „Hallo Welt“ zeigt, wie die speziell für Oberflächenbeschreibungen bereitgestellte Informationsstruktur mit der Beschreibungssprache Lucia ausgefüllt wird.

Im Projektalltag können Sie die Sprachbeschreibung und die Beispielmodelle als Referenzmaterial verwenden. Sie finden in der Sprachbeschreibung Formulierungsbeispiele für typische User Interface-Bedienungssituationen und Lösungsvorschläge für übliche UI-Spezifikationsprobleme.

UI-Beschreibung wie Quellcode verwenden

Kapitel 7: „***Architektur***“, in dem die UI-Beschreibung zu einem Entwicklungsartefakt gekürt und gezeigt wird, was Entwickler daraus machen können.

Eine UI-Spezifikation, die in Form und Inhalt eindeutig ist, lässt sich auch maschinell auswerten und zum ausführbaren Code umwandeln. Damit lässt sich eine Spezifikation ähnlich wie Quellcode in einer höheren Programmiersprache verwenden.

Die in der formalen Spezifikation enthaltenen Informationen können maschinell ausgewertet werden und daraus automatisiert zum Beispiel der Programmcode für Dialogseiten der Anwendung erzeugt werden.

UI in der Architektur verankern

Dieses Kapitel zeigt auf, wie man die modellgetriebene Architektur (MDA) auf UI-Beschreibungen anwenden kann und welche Produktivitätszuwächse damit möglich werden. Durch Transformationsalgorithmen kann der Spezifikationsstand zum Beispiel als lauffähiger Prototyp der Oberfläche bereitgestellt und so schon vor der Implementierung geprüft werden. Integriert man solche Transformationsalgorithmen in die Architektur der Anwendung, dann kann die Umsetzung der Oberflächenbeschreibung in die ablauffähige Oberfläche drastisch verkürzt werden.

Das eröffnet für Entwickler und Anwendervertreter neue Möglichkeiten, die anhand von Praxisbeispielen erläutert werden.

UI in die Modellierung der Funktionalität einbinden

Kapitel 8: „***Funktionsmodellierung***“, in dem UI-Modelle und UML-Modelle (z.B. Use Cases, Klassen, Abläufe) zusammen angewandt werden, sich ergänzen und gemeinsam wirken.

Das UI beeinflusst die Funktionen und die internen Abläufe der Software. Umgekehrt beschränken Funktionen des technischen Softwaresystems die Möglichkeiten des User Interface. Die UI-Modellierung verzahnt sich daher eng mit der Modellierung der internen Strukturen und Abläufe in der übrigen Software.

Das Kapitel zeigt, wie Informationen aus UI-Modellen in UML-Modelle einfließen und dort weiterverwendet werden können. Ebenso wird aufgezeigt, wie UI-Modellierung im Kontext bereits vorhandener Funktions- und Ablaufmodelle erfolgreich und einander ergänzend durchgeführt werden kann.

Funktionalität anhand der UI-Beschreibung ableiten und prüfen

Die Verbindung von gängigen Modellierungs- und Konzeptionsmethoden mit der UI-Modellierung führt zu einem ausgewogenen Ganzen. Entwickler finden in diesem Kapitel eine Praxisanleitung zur Ergänzung der gängigen Entwurfsmethoden mit UI-Modellen. Nutzen: UI- und Business-Logik sind von Anfang an aufeinander abgestimmt; benötigte Funktionen und Datenstrukturen werden frühzeitig erkannt. Die laufende Vollständigkeit und Korrektheit der Funktions- und Datenentwürfe wird besser abgesichert.

UI-Spezifikation als Steuerungsdaten für Prozesse übernehmen

Kapitel 9: „***Prozesssteuerung***“, in dem die in einer UI-Spezifikation enthaltenen Informationen zu Ablaufgraphen und zu elektronischen Workflows destilliert werden.

Eine UI-Beschreibung enthält Informationen über Dialogabfolgen und Details über die Reihenfolge der Tätigkeiten beim Ausführen von Aufgaben mit der Software.

Workflow aus UI-Beschreibung ableiten

Diese Informationen können herausgefiltert und zur Steuerung von Arbeitsschritten auf Workflow- und Prozessebene herangezogen werden.

Für Entwickler von prozessorientierten bzw. workflowgesteuerten Anwendungen (zum Beispiel Sachbearbeitungs- und Verwaltungssoftware) zeigt das Kapitel Möglichkeiten auf, Anwendungsworkflows regelbasiert anhand der aus der UI-Beschreibung gewonnenen Ablaufschritten und Regeln mit Steuerungsdaten zu versorgen.

Benutzer- und Entwicklungs-Dokumentation generieren

Kapitel 10: „***Dokumentation***“, in dem die in einer UI-Beschreibung gesammelten Informationen als Gerüst für Handbücher und Hilfesysteme dienen.

Eine UI-Beschreibung enthält vielfältige Informationen über Wertebereiche, Abläufe, Dialoginhalte und Fallabhängigkeiten. Zum sauberen Arbeiten gehört es auch, für einzelne Oberflächenbe-

standteile deren Zweck sowie die zugrunde liegenden Anforderungen in der UI-Beschreibung zu dokumentieren. Aus diesem Informationsgefüge können vielfältige Dokumente und Auswertungen generiert werden.

Von großem Nutzen für Dokumentationsentwickler ist die Möglichkeit, das Gerüst der Dokumentation jederzeit aus der aktuellen UI-Beschreibung generieren zu können. So können alle bereits vorhandenen Informationen frühzeitig verwertet werden.

Zusammen mit der Möglichkeit, die UI-Spezifikation als Oberflächenprototyp ausführen zu können, kann die Benutzerdokumentation parallel zum Entwurf und zur Realisierung der Anwendung erstellt werden. Damit lässt sich sowohl die Qualität des Gesamtprodukts steigern als auch die Lieferzeit verkürzen.

Automatisiert gegen UI-Beschreibung testen

Kapitel 11: „***Testen***“, in dem die Informationen in der UI-Beschreibung als Quelle für automatisiert generierte Testszenarien in Szene gesetzt werden.

Verwerten der UI-Beschreibung beim Testen der Anwendung

Eine UI-Beschreibung enthält Informationen über Wertebereiche für Eingaben und Details über bedingte Reaktionen auf Eingaben. Aus diesen Informationen können automatisiert synthetische Testfälle über zufällige Abfolgen von Eingaben generiert werden und gegen die erwarteten Reaktionen geprüft werden. Auf diese Weise kann die UI-Beschreibung beim Testen der Anwendung verwertet werden.

Die in diesem Kapitel vorgestellten Methoden sind in der Umsetzungsphase und in der Testphase (Construction, Transition) von Nutzen.

Projekt mit der UI-Beschreibung steuern

Kapitel 12: „***Projektmanagement***“, in dem UI-Modelle in die bei einem Projekt üblichen Abnahme-Dokumente (z.B. Grobkonzept, Fachkonzept, Implementierungsspezifikation) eingeordnet werden.

Einbinden der UI-Entwicklung in das Projektmanagement

Genauso wie UML-Modelle und Konzeptdokumente müssen UI-Modelle klar und nachvollziehbar in das Projektvorgehen eingegliedert sein. Da UI-Modelle aufgrund ihrer inhaltlichen Breite Überschneidungen mit beinahe allen Entwicklerrollen haben, ist ein besonders intensives und integrierendes Management der UI-Beschreibung selbst und ihrer Wechselwirkungen mit anderen Entwicklungsbereichen unabdingbar.

Das Kapitel zeigt auf, wie das Erstellen des Artefakts User Interface im Projekt installiert wird und wie dabei die Akzeptanz und Mitwirkung durch die einzelnen Entwicklerrollen erreicht wird.

Die UI-Modellierung ist ein Querschnittsthema und muss daher aktiv und integrierend in allen Projektphasen verankert sein, um

von den Beteiligten als nutzbringend für die eigene Arbeit erkannt und somit mitgetragen zu werden.

Sie finden in diesem Kapitel Tipps und Vorgehensvorschläge für die erfolgreiche Einführung, nutzbringende Anwendung und nachhaltige Integration der UI-Architektur in Softwareprojekten.

Das Kapitel ist auch für Entwickler und Architekten, nicht nur für Projektleiter interessant. Es zeigt an Fallbeispielen mögliche Risiken (z.B. falsche Erwartungen an die UI-Architektur, Kompetenzverschiebungsängste, Abwehrhaltungen) und Möglichkeiten zur Vermeidung dieser Risiken.

UI-Beschreibung wieder verwenden

Kapitel 13: „***Lebenszyklusmanagement***“, in dem der Generationenwechsel der Technik die Hauptrolle spielt. Eine plattformunabhängige Oberflächenbeschreibung überbrückt den Technologiewechsel.

Evolutionäres Cross-Plattform-Management

Oberflächen müssen nicht für jede neue Technologieplattform von Grund auf neu entwickelt werden, sondern können über Technologiewechsel hinweg wieder verwendet und weiterentwickelt werden.

Die in diesem Kapitel vorgestellten Ansätze sind bei der Planung von Releases, Entwicklungsstufen und von Reengineering-Projekten hilfreich.

Ebenso liefert dieses Kapitel eine Zusammenfassung der in der UI-Architektur angesprochenen Themengebiete und einen Ausblick auf die weiteren Möglichkeiten und Chancen der konsequenten Anwendung der UI-Architektur im Projektalltag.

Geplante Online-Begleitmaterialien zum Buch

Begleitmaterial Online auf www.Benutzeroberflaeche.de:

- Beispiele von User-Interface Modellen in der User Interface-Beschreibungssprache Lucia,
- Eine detaillierte Grammatik von Lucia,
- Eine Demonstration des Ausleitens von Prototypen aus UI-Spezifikationen durch ein Web-Service,
- Eine detaillierte Übersicht über aktuelle Werkzeuge zur UI-Entwicklung.

1.7 Lesen von UML-Diagrammen

Weil zur Erklärung von Zusammenhängen im Buch viele UML-Diagramme verwendet werden, folgt eine kurze Einführung in UML-Klassendiagramme und UML Use Case-Diagramme.

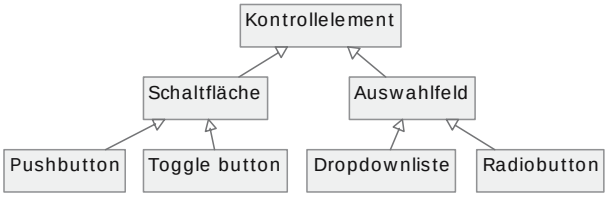
Diese Kurzeinführung ist für die Leser bestimmt, die mit der UML-Notation nicht vertraut sind. Für die UML-Kundigen sind in

den Einführungs-Diagrammen einige Paradigmen der UI-orientierten Architektur enthalten.

UML- Klassendiagramm

- Klasse
  - Definiert einen Begriff eines Fachvokabulars durch seine eindeutige Klassifizierung, z.B. Benennung einer Klasse „*Kontrollelement*“.

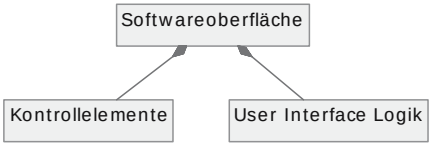
- Spezialisierung bzw. Generalisierung
  - Bringt Klassen in die Beziehung „ist ein...“, z.B. „ein Pushbutton *ist* ein Kontrollelement“.



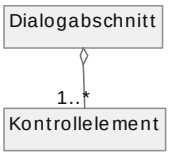
Eine detaillierte Aufzählung der Kontrollelemente in einem UI finden Sie im Kapitel „Oberfläche“

  - Anmerkung: Ein User Interface enthält neben Schaltflächen und Auswahlfeldern natürlich noch andere Kontrollelemente, z.B. Text-Eingabe-Felder, Text-Ausgabe-Felder, Grafiken, Tabellen, usw.

- Containment / Komposition / Aggregation
  - Bringt zwei Klassen in die Beziehung „ist ein Bestandteil von ...“ bzw. „besteht aus ...“ oder „enthält“, zum Beispiel „Eine Softwareoberfläche *besteht* aus Kontrollelementen und aus UI-Logik“.



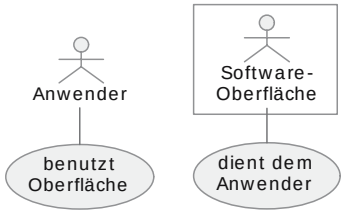
- Multiplizität
  - Gibt an, wie viele Exemplare der durch eine Containmentbeziehung (oder durch eine einfache Assoziation) verbundenen Klassen gemeint sind, z.B. „Ein Dialogabschnitt enthält *1..** Kontrollelemente“.



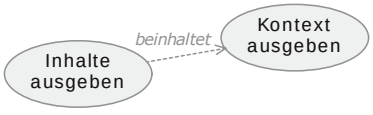
- Abhängigkeit / Verwendung
  - Bringt zwei Klassen in eine Abhängigkeitsbeziehung, z.B. „Ablaufnetz *verwendet* Dialogseiten“ oder „Interaktion benötigt (benutzt/verwendet) Kontext“.

UML Use Case-Diagramm

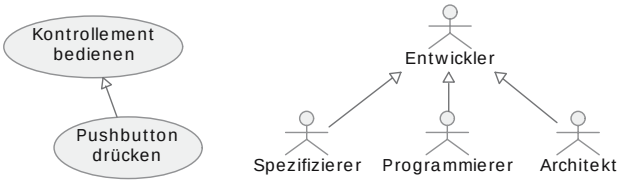
- Akteure („Handelnde“)
  - Definiert eine agierende Anwenderrolle. Aktoren können sowohl Menschen als auch Systeme sein.
- Verwendungsfälle / Anwendungsfälle
  - Sind Tätigkeiten, die der Akteur mit Hilfe des verwendeten Systems (hier also mit einer Softwareanwendung) durchführen kann.



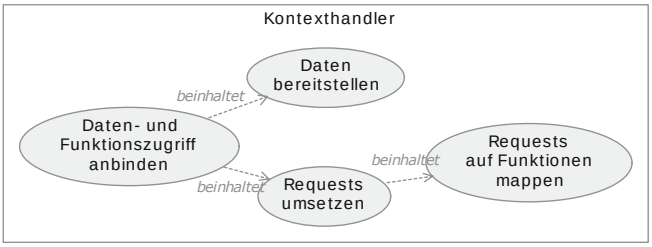
- Abhängigkeit / Verwendung
  - Bringt Anwendungsfälle in eine Abhängigkeitsbeziehung, z.B. „Inhalte ausgeben *beinhaltet* Kontext ausgeben“ oder „Prioritäten berücksichtigen *erweitert* Signale interpretieren“.



- Spezialisierung / Generalisierung
  - Bringt Anwendungsfälle oder Aktoren in die Beziehung „ist ein...“, z.B. „Pushbutton drücken *ist* Kontrollelement bedienen“ oder „Spezifizierer *ist* ein Entwickler“.



- Systemgrenzen
  - Kennzeichnen mehrere Anwendungsfälle als zu einem System zugehörig, z.B. „Die Anwendungsfälle ‚Daten bereitstellen‘ und ‚Requests auf Funktionen mappen‘ werden *im* Kontexthandler bereitgestellt“.



Die obige Aufstellung zeigt nur einen Auszug aus den vielfältigen Möglichkeiten, die Klassen- und Use Case-Diagramme bieten. Eine weitgehend vollständige Beschreibung der UML finden Sie z.B. in [Jeckle 2004].

1.8 Zusammenfassung

Bauentwurfslehre für Oberflächen

Oberflächen sind von den ursprünglich kargen Kommandokonsolen zu multimedialen Assistenten für die eigentliche Funktionalität einer Software gewachsen.

In dialogintensiven Anwendungen machen Präsentation und Interaktion zwischen Anwender und Anwendungsoberfläche oft mehr als 2/3 der Software aus [Myers 1992]. Die eigentliche Transaktion kommt hingegen erst am Schluss der Verhandlung zwischen Anwender und User Interface zur Ausführung.

Der Anteil der Geschäftslogik am Gesamtumfang einer dialogintensiven Anwendung ist kleiner, als bislang aus der Sicht von Objekt- und Ablaufmodellen angenommen wird.

Es lohnt sich also, Modelle, Methoden und Werkzeuge zur formalen Beschreibung von Oberflächen zu entwickeln.

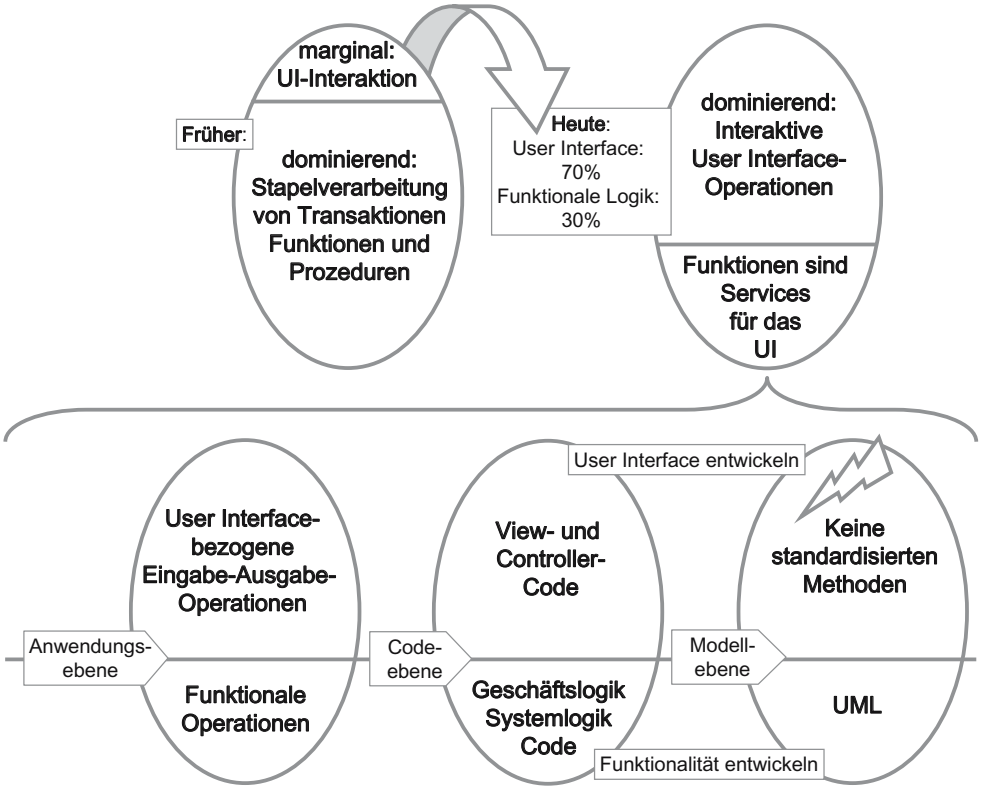


Bild 13: User Interfaces machen den Großteil einer Software aus

User Interface-orientierte Architektur ist eine Bauentwurfslehre für durchdachte, interaktive Softwareoberflächen und ein Leitfaden für erlebbare und eindeutige User Interface-Spezifikationen.

Das Buch behält Funktionalität und Look-and-Feel der Software gleichermaßen im Blick: Der Fokus liegt auf dem Verstehen, Planen und Umsetzen der für das ***Verwenden*** der Software geforderten Eigenschaften.

UI-Architektur versus GUI-Design

Der Unterschied zwischen User Interface-Architektur und den vielen auf dem Markt verfügbaren GUI Design-Werkzeugen ist die Fokussierung auf die ***Informationsarchitektur*** und auf das methodische Vorgehen.

GUI-Bücher stellen Regeln für die Gestaltung von Oberflächen auf. Die User Interface-Architektur geht hingegen auf den Informationsgehalt ein, der zum Bau einer Oberfläche notwendig ist und darauf, wie man diese Informationen formuliert, d.h. mit welcher Notation, in welchem Prozess und mit welchen Werkzeugen.

UI-Architektur von der herkömmlichen Softwarearchitektur differenzieren

Die Unterschiede der User Interface-orientierten Architektur zur herkömmlichen Softwareentwicklung sind:

- Die Entwickler entwickeln das User Interface als eigenes Artefakt. Heute wird das UI nicht als eigenes Artefakt verstanden.
- Nach dem Ansatz der UI-Orientierung werden die funktionalen Anforderungen aus den UI-Anforderungen heraus entwickelt. Heute werden UI-Anforderungen nachrangig zu den funktionalen Anforderungen erhoben.

Das Einführungskapitel hat Ihnen einen Überblick über den heutigen Stand der User Interface-Entwicklung, über den Begriff der UI-Orientierung sowie über die typischen Fragestellungen und Lösungsansätze für die Herausforderungen bei der Entwicklung dialogintensiver Anwendungen gegeben.

Mit dem Wissen aus dem Einführungskapitel verfügen Sie über einen Leitfaden zu den einzelnen Themen der UI-Architektur und können den Nutzen der UI-Orientierung für Ihre Softwareprojekte überschlagen.

Aber wie funktioniert eigentlich eine Softwareoberfläche und wie hängt das alles, was auf dem Bildschirm zu sehen ist und das, was die Anwendung macht, zusammen? Ist in Anbetracht der Vielfalt verschiedener Oberflächentypen und anwendungsspezifischer Eigenarten eine strukturierte und exakte Form der Beschreibung von Softwareoberflächen möglich?

Um die Antworten auf diese Fragen geht es im nächsten Kapitel.

Kapitel 2: Oberfläche

… in dem die Struktur und die Bestandteile einer Benutzeroberfläche unter die Lupe genommen werden und eine allgemeine Gliederung für User Interfaces aufgestellt wird.

Kapitelziele

- *Wissen, was eine Softwareoberfläche ist.*
- *Wissen, woraus eine Softwareoberfläche besteht.*
- *Gliederung der Bestandteile einer Oberfläche.*

Umgangssprachlich werden die vom Anwender wahrnehmbaren Teile der Softwareanwendung mit vielen synonymen Bezeichnungen bedacht. Üblich sind Bezeichnungen wie Benutzerschnittstelle, Bedieneroberfläche, Anwendungsoberfläche, Softwareoberfläche, Dialogmasken oder Dialogschnittstelle. In Spezialistenkreisen werden weiter differenzierte Fachbezeichnungen verwendet: Präsentationsschicht, UI (User Interface), GUI (Graphical User Interface), Human Computer Interface (HCI) oder Mensch-Maschine-Schnittstelle (MMS).

In diesem Kapitel machen wir das, was Ingenieure am liebsten tun: Wir nehmen uns einen Schraubenzieher, schrauben den Deckel ab und gucken gründlich und tief in die Zahnräder eines User Interfaces hinein.

Eine Softwareoberfläche ist kein Monolith und auch keine Verpackung um die Softwarefunktionen, sondern eine Maschine mit verschiedenen Teilen, die gemeinsam eine komplexe Vermittlungsaufgabe erfüllen.

Bild 14: Eine Softwareoberfläche besteht aus verschiedenen Teilen

Eine ***Softwareoberfläche*** ist eine ***komplexe*** Maschine, die zwischen Anwender und Softwarefunktionen ***vermittelt***.

Üblich ist bei heutigen Oberflächen auch, dass sie den Anwender anleiten. Dazu gehören entsprechender Aufbau der Dialoge selbst, aber auch kontextsensitive Hilfefunktionen und On-Screen-Hilfe. ***Benutzerdokumentation*** kann aus diesem Blickwinkel als Teil der Softwareoberfläche gesehen werden.

Dieses Kapitel zeigt, welche verschiedenen Bestandteile eine Oberfläche hat und wie diese gegliedert werden können. Die Bestandaufnahme liefert einen Überblick über Zusammenhänge und Unterschiede zwischen den verschiedenen Teilen einer Softwareoberfläche.

Das Ergebnis dieser ***informationellen Analyse*** eines User Interface ist ein ***Gliederungsgerüst*** für das Beschreiben der Benutzerschnittstellen von Softwareanwendungen.

Zu dieser Grundgliederung gehören Abläufe und Inhalte sowie Wechselwirkungen und Beziehungen zwischen den Oberflächenbestandteilen, Anwendungsfunktionen und Anforderungen.

UI-Perspektiven

Auf diese Strukturierungsgrundlage, die ich ***UI-Perspektiven*** nenne, wird die User Interface-Architektur in allen Kapiteln zurückgreifen und aufbauen. Sie können lernen, mit diesen Perspektiven in unterschiedlichen Phasen des Softwareprojekts umzugehen und so verschiedene Aspekte der Oberfläche zu sehen und zu entwerfen.

Eine UI-Gliederung nach UI-Perspektiven kann zum Beispiel zum

- Vergleichen des Informationsgehalts von verschieden umgesetzten Oberflächen oder zum
- Bewerten der Vollständigkeit einer zur Prüfung vorgelegten Oberflächenbeschreibung

herangezogen werden.

Sie können die UI-Perspektiven auch verwenden, um

- oberflächenrelevante Informationen in einen Gesamtzusammenhang einzuordnen und die
- UI-Beschreibung schrittweise von der ersten Skizze bis zu einer umsetzungsreifen Detailspezifikation der Softwareoberfläche zu ergänzen.

2.1 Historische Entwicklung der Softwareoberflächen

Die Softwarekrise war der Motor für systematisches Entwickeln von Software

Seit den Anfängen der Softwareentwicklung gibt es gute und schlechte Softwareanwendungen und damit gute und schlechte Benutzeroberflächen. Ende der 60er Jahre hat die so genannte ***Softwarekrise*** die Entwicklung von ingenieurmäßigen Methoden und Vorgehensmodellen für den Softwarebau eingeleitet. Die Softwareindustrie begann, analog zur Bau- und Automobilindustrie, Software systematisch zu planen und zu entwickeln.

Die so begonnene kontinuierliche Auseinandersetzung mit den Schwächen der Methoden und Werkzeuge der Softwareentwicklung hat zu einer Vielzahl nützlicher Hilfsmittel geführt, z.B.

- SADT (Structured Analysis and Design Technique),
- IDEF (Integrated Computer Aided Manufacturing Definition Languages),
- ERM (Entity Relationship Model),
- OOD (Object Oriented Design),
- UML (Unified Modelling Language) oder
- XML (Extensible Markup Language).

Die Verbreitung und die fortschreitende Standardisierung dieser Hilfsmittel und Methoden helfen, Kommunikationsbrüche in Prozessen der Softwareentwicklung zu vermeiden und zu schließen.

Benutzeroberflächen sind bis heute nur nachrangig (XForms, XUL, UIML), in diese Methodenentwicklung eingeschlossen.

Von Stapelverarbeitung zu Dialogtransaktionen

In prozeduralen Anwendungen gibt es keinen Dialog mit dem Anwender

In der Batchwelt der frühen EDV kamen bis in die späten 50er Jahre Anwendungsoberflächen mit interaktiven Dialogseiten gar nicht vor. Prozedurale Anwendungen erledigten die Verarbeitung von gestapelten Eingabedaten ohne Anwenderinteraktion und lieferten die Ergebnisse meist in Form von Listen oder neuen Datenkartenstapeln. Tastatur und Bildschirm wurden 1960 zum ersten Mal an einen Computer angeschlossen und lösten unter den Großrechnerherstellern Panik aus [Rose 1985, S.48].

Bei dialogtransaktionalen Oberflächen wird der Anwender am Fluss der Transaktionen entlang geführt

Mit dem Aufkommen der ersten dialogisierten transaktionalen Anwendungen in den frühen 60ern änderte sich an der Bedienung der Softwareprogramme zunächst wenig, da hinter den Dialogseiten meist die bisherigen Batchaufrufe maskiert wurden. In den Masken wurden die Parameter für den Stapellauf eingetragen und mit der Maskenfreigabe an die Prozedur übergeben. Oberflächen ergaben sich damit „von selbst“ aus den Prozedu-

ren. Als Abfallprodukt des Funktionsdesigns stellte die Anwendungsoberfläche keine eigenen Ansprüche an die Entwicklung von Methoden zu ihrer systematischen Entwicklung.

Mit der Zunahme von Dialogtransaktionen wurden Maskenbeschreibungsformalismen entwickelt und in die Programmiersprachen und Entwicklungswerkzeuge integriert, z.B. SDD (Structured Dialog Design) in IBMs RPG (Report Program Generator), Input- und Display-Befehle in COBOL (Common Business Oriented Language) oder IO-Makros in Assembler auf BS2000-Systemen.

Später folgten diesem Ansatz fortgeschrittene Formen von Maskengeneratoren in dBase, FoxPro, in verschiedenen RDBMS (Relational Database Management System) wie Oracle oder DB/2 und in anderen integrierten Werkzeugen zur Entwicklung von datenorientierten Geschäftsanwendungen.

Grafische Benutzeroberflächen

Softwareoberflächen veränderten sich stark mit dem Markteintritt der Computermaus und mit dem Erfolg grafischer Betriebssysteme (bzw. Betriebssystemaufsätze) wie

- Mac OS,
- X-Windows,
- GEM (Graphical Environment Manager),
- GEOS (Graphic Environment Operating System),
- MS Windows (Microsofts Antwort auf GEM) oder
- KDE (K Desktop Environment).

Grafische User Interfaces arbeiten ereignisgesteuert

Durch GUIs (Graphical User Interfaces) wurden Anwendungen zunehmend reaktiv. Die wesentliche Neuerung der grafischen Oberflächen zu bisherigen Dialogbildschirmen ist die ***ereignisorientierte*** Arbeitsweise.

In transaktionsorientierten Dialoganwendungen haben Oberflächen an bestimmten Stellen im Programmablauf bestimmte Eingaben des Benutzers gefordert. Mit der Einführung der Ereignisorientierung wurde diese strikte Führung des Benutzers gelockert. Der Benutzer kann die angebotenen Bedienungselemente in beliebiger Reihenfolge verwenden. Die Anwendungsoberfläche reagiert auf diese Verwendungsereignisse. In gleicher Weise kann die Anwendungsoberfläche auf anwendungsinterne Ereignisse reagieren. Eine kompakte Beschreibung der ereignisgesteuerten Programmierung finden Sie z.B. in [Lauer 2002, S.22ff].

Bei grafischen Oberflächen verflochten sich Transaktionen mit den für den Benutzer sichtbaren Steuerungs- und Rückkopplungselementen. Auf Entwicklungsseite wurden die Werkzeugkästen der bisherigen Maskengeneratoren um grafische Kontrollelemente (GCI, Graphical Control Items) und um Ereignisbehandlung erweitert. Es wurden grafische SDKs (Software Development Kits) und GCLs (Graphical Control Libraries) entwickelt.

Die Entwickler begannen, ***Transaktionen und Transaktionsteile an einzelne Kontrollelemente zu koppeln***. Die Transaktionen wurden feinkörniger, um die neuen Kontrollelemente mit Statusinformationen und Wertebereichen beliefern zu können. Zum Beispiel benötigte man jetzt statt der Eingabe einer Artikelgruppe in einem Schlüsselfeld eine Combobox mit Auswahl der Artikelgruppe und somit die neue Transaktion „hole die Namen aller Artikelgruppen".

Enge Kopplung zwischen Transaktionen und Kontrollelementen

Es wurden immer vielseitigere Kontrollelemente entwickelt und ihre Interaktionen immer dichter an den Datenhaushalt der Anwendungen gekoppelt. Dabei umgingen die Entwickler gerne die hinderliche Transaktionsschicht. Die so entstandenen Anwendungen, deren Hauptmerkmal eine hochinteraktive Benutzeroberfläche mit sehr feinkörnigen Interaktionen ist, werden als ***Fat Clients*** bezeichnet.

Trotz der hohen Komplexität der in einer grafischen Oberfläche auftretenden Wechselwirkungen von Arbeitsabläufen, Darstellung, Eingaben und Transaktionen blieb die Methodik der UI-Entwicklung zunächst ein Randthema.

Auch bei Fat Client GUIs wird die Entwicklung der Oberflächen oft weiter als Abfallprodukt der Entwicklung von Funktionalitäten angesehen.

Von Fat Clients zu Thin Clients

Die Anwendungsentwickler haben sich erfolgreich mit grafischen Oberflächen und mit der Ereignisorientierung arrangiert. Das Paradigma vom Aufsetzen der Oberfläche als „Masken" über Funktionen und anwendungsinterne Abläufe wurde beibehalten.

Durch die 1991 begonnene weltweite Verbreitung und kommerzielle Nutzung des WWW (World Wide Web) und damit der HTML (Hyper Text Markup Language) hat sich die Funktions- und Bauweise von Softwareoberflächen - zunächst langsam und weitgehend unbemerkt - abermals dramatisch verändert.

Anfangs diente die HTML zur standardisierten Darstellung wissenschaftlicher Texte. Die Kommunikation zwischen UI (Web Browser) und Anwendungs-Backend (Web Server) erfolgte zu-

nächst nur auf Ebene der Web-Seiten über HTTP (Hyper Text Transfer Protocol). Die Möglichkeit, Anwendungen anzusteuern und damit dynamische Inhalte darzustellen, kam später dazu, zunächst als CGI (Common Gateway Interface). Bei CGI wurde sozusagen die Ereignisverarbeitungsschleife eines Fat Clients als Serverprogramm nachempfunden. Danach folgten viele weitere Möglichkeiten für interaktive Web-Anwendungen, z.B. Applets, Servlets oder Java Server Pages (JSP).

Der große Unterschied zwischen einer auf HTML und Web Browser basierten und einer vom Fat Client gewohnten Anwendungsoberfläche ist, dass nach dem HTTP-Konzept die Grenze zwischen Transaktionen und Präsentation hart, weil physisch, ist. Die im Fat Client üblichen ***Intensivkopplungen zwischen einzelnen Controls und den feinkörnigen Transaktionen*** sind bei einem Thin Client teuer, weil für jeden Kopplungszyklus eine Servertransaktion mit Datenübertragung notwendig ist.

Thin Web Clients haben Ähnlichkeiten mit dialog-transaktionalen Oberflächen, in der man den Zentralrechner hinter der Bildschirmmaske spüren kann. Für Anwender, die sich an die beliebig interaktive Fat Client-Welt gewöhnt hatten, in der man das Gefühl haben konnte, dass man die Anwendung mit allen Daten zur alleinigen Nutzung auf seinem Rechner hat, kann sich das wie ein Rückschritt anfühlen.

Natürlich schränken Thin Clients die unbegrenzten Kopplungsmöglichkeiten zwischen Teiltransaktionen und Controls der Fat Clients ein. Und natürlich möchten die Anwender nicht eingeschränkt werden.

Das Web zeigt die Grenze zwischen Oberfläche und Funktionalität auf.

Das Web hat die Grenze zwischen der Präsentationsschicht und der Transaktionsschicht wieder spürbar gemacht. Dadurch ist auch sichtbar geworden, wieviel von der Anwendung potentiell Präsentationsschicht ist. Damit ist auch spürbar geworden, welche komplexen IO- und Interpretationsvorgänge in einer Oberfläche enthalten sind.

Das Web bricht Grenzen zwischen Anwendungen auf.

Das Web durchbricht aber auch die Grenzen zwischen Einzelanwendungen und ermöglicht die Fokussierung auf Prozesse und Workflows. Anwendungen werden über das Web global miteinander vernetzt.

Von Thin Clients zur SOA und weiter zur User Interface-orientierten Softwarearchitektur

Die Welten der (zentralen und funktionsorientierten) Transaktionen und der (lokalen und anwenderorientierten) Interaktionen mit ihren jeweiligen Vorteilen und Wechselwirkungen sind durch Web-Oberflächen in ein produktives Spannungsfeld getreten.

Die Entwickler von Web-Anwendungen arbeiten zum Beispiel mit .Net und Active-X, DHTML und JSP oder mit ZOPE und Python zwar in verschiedenen Lagern, doch zieht man dabei durchaus am gleichen Strang: Nämlich wieder den Komfort der Fat Client Verhältnisse einzuführen.

Die Bemühungen der Softwarehersteller zielen darauf ab, das aus der Fat Client-Welt gewohnte Interaktionslevel in die Web Client-Welt zu übertragen. Dies ist an sich kein systematisches Problem, sofern das Netz genügend Bandbreite zur Verfügung stellt.

Wenn jedoch zum Beispiel durch .Net die Grenze zwischen Transaktion und Präsentation wieder weich wird, könnte das negative Folgen haben. Eine nicht mehr beherrschbare, weil nicht exakt formulierbare Vermengung von Präsentations-, Prozess-, Anwendungs- und Geschäftslogik könnte ein systematisches Problem sein.

Die Service-orientierte Architektur (SOA) mit Web Services strebt die Bereitstellung von Funktionalitäten und Prozessbausteinen als Dienste an. Web Services ermöglichen das Bereitstellen von verteilten Funktionalitäten, die über XML-Schnittstellen angesprochen und damit leicht in Web User Interfaces eingebunden werden können.

Auf der Systemseite werden die Möglichkeiten der Aufbereitung und Darstellung von Informationen, auf der Anwenderseite die Möglichkeiten, mit diesen Informationen zu interagieren, komplexer [Khazaeli 2005].

Die Antwort auf die Frage, was sich eigentlich an der Anwendungsoberfläche abspielt und wie sich das mit der Transaktionslogik verzahnt, hat im Kontext der immer komplexer werdenden Anwendungen und der neuen technischen Möglichkeiten des Web eine neue Dringlichkeit erhalten.

Mit geeigneten User Interface-Modellen, die über das Aufsetzen von Masken auf Funktionen hinausgehen, kann das Risiko einer neuen komplexitätsbedingten Krise in der Softwareentwicklung begrenzt werden.

2.2 Oberfläche gliedern

Wer den Auftrag hat, eine Softwareoberfläche zu erstellen, muss Verschiedenes über die für ein ordentliches UI benötigten Dialogabläufe, Ablaufschritte, Kontrollelemente und Interaktionen in Erfahrung bringen.

Unabhängig davon, ob die Oberfläche für eine Kundenverwaltung, für ein Navigationssystem , für ein Verwaltungsprogramm oder für eine Vertriebsunterstützungssoftware ist, kann man die

Aufgabenstellung nach den Merkmalen der Softwareoberfläche analysieren und thematisch gliedern.

Die im Folgenden (in Form von Leitfragen) beschriebenen Klassifizierungsmerkmale können als Anleitung für eine UI-orientierte Anforderungsanalyse verwendet werden.

Gliedern der UI-Beschreibung mit Leitfragen

Leitfragen der UI-Beschreibung

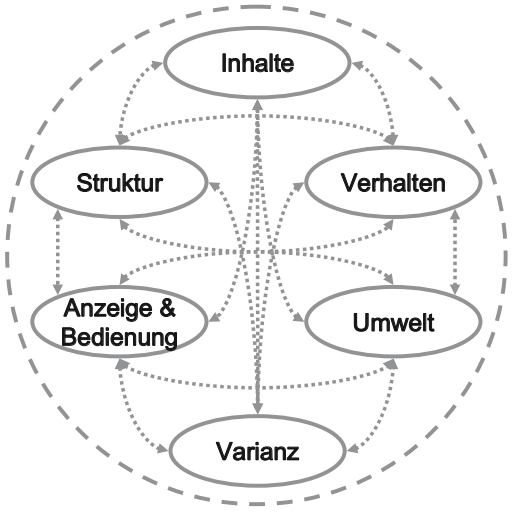
Um eine Anwendungsoberfläche entwerfen zu können, stellt der Entwickler bzw. der Analytiker eine Menge Fragen. Die Fragen können das generelle Aussehen, Verhalten und Funktion der Anwendung, einzelne Dialogseiten, Buttons oder Eingabefelder oder den Dialog zwischen Anwender und Oberfläche in speziellen Situationen betreffen. Sowohl die Fragen als auch die Antworten können schnell ein komplexes und schwer zu überblickendes Informationsgemenge bilden.

Die Gliederung der Fragen in merkmalbezogene Themen kann helfen, die Antworten in eine strukturierte Form zu bringen und so den Überblick über eine UI-Beschreibung zu behalten.

Grobgliederung für eine UI-Beschreibung

Die beim Entwickeln eines User Interface aufkommenden Fragenkomplexe lassen sich wie folgt zu einer Grobgliederung für eine UI-Beschreibung formen.

Bild 15 Themengliederung für das Beschreiben einer Oberfläche



Die obige Abbildung zeigt, dass sich die Beschreibung eines User Interface in sechs verschiedene Themen gliedern lässt und dass die verschiedenen Themen einer UI-Beschreibung miteinander zusammenhängen.

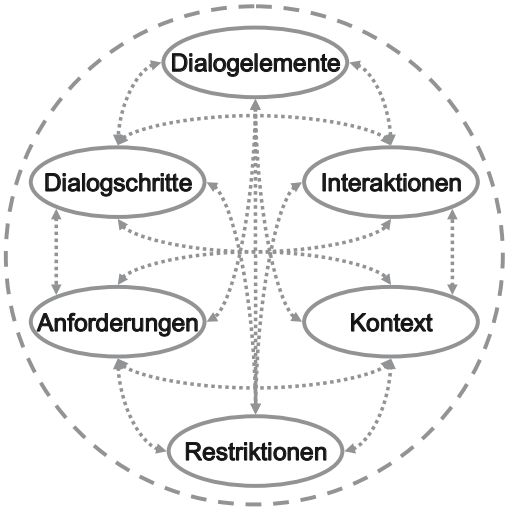


Bild 16: Informationsgeflecht einer UI-Beschreibung

Die obige Abbildung zeigt das Informationsgeflecht, das in der Themengliederung enthalten ist und in einer UI-Beschreibung wiedergegeben wird.

Inhalte / Dialogelemente

Gegenstand der Anwendung, Funktionalität, Ablaufsequenzen

- Was beinhalten die Abläufe und Inhalte der Anwendung aus fachlicher Sicht?
  - Standard-Dialogabfolgen (zuerst Normalfälle)
  - Inhalte von Dialogmasken (zuerst Hauptelemente)
- Wodurch sind diese Abläufe und Inhalte gekennzeichnet?
  - Varianz und Vermischung von Dialogabläufen
  - Maskenaufbau, typische Anwender-Interaktionen

Struktur / Dialogschritte

Ablaufgliederung, Anwendungsaufbau, Ablaufzusammenhänge

- Wie hängen die Aktivitäten und ihre Inhalte strukturell in der Anwendung zusammen?
  - Gliederung der Tätigkeiten
  - Zuordnung von Dialogmasken zu Ablaufschritten
  - Tätigkeitsauswahl
  - Wiederholung und Unterbrechungen
  - Datenorganisation

Anzeige und Bedienung / Anforderungen

Darstellung, Bedienungssystematik, Anzeige-, Bedien- und Navigationsregeln

- Wie wird eine konsistente Darstellung der Anwendungsinhalte erreicht?
- Wie wird der Anwender durch den Prozess und die Arbeitsschritte geführt?
  - Navigationselemente
  - Layouts
  - Widgets

Verhalten / Interaktionen

Präsentationslogik, Schnittstelle zur Geschäftslogik

- Nach welchen Regeln erfolgt die Steuerung der Anwendungsabläufe?
- Wie wird die Dynamik der Abläufe zwischen den Arbeitsschritten und innerhalb der Arbeitsschritte kontrolliert?
  - Ereignisse
  - Auswerten von Situationen
  - Reaktionen
  - Dialogfortschritt

Umwelt / Kontext

Skalierung, Präsentations- und Funktionskontext

- Wie werden die verschiedenen Sichten (rollen- und bearbeitungsstandbedingt) auf den gleichen Anwendungsinhalt abgebildet?
- Wie reagiert die Oberfläche auf sich ändernde Umweltbedingungen?
- Wie werden in Abhängigkeit vom Bearbeitungsstand die auszuführenden Funktionen und Funktionsparameter bestimmt?
  - Ermitteln von Situationen
  - Verwenden von Anwendungsfunktionen
  - Verwenden von Anwendungsdaten

Varianz / Restriktionen

Umgebung, Varianten, Mandanten, Zulieferer

- Wie werden mandanten- und versionsabhängige Varianten und Abweichungen der Anwendungsabläufe und deren Inhalte verwaltet?
  - Länder- und Sprachabhängige Ressourcen
  - Rollenspezifische Unterschiede in UI-Elementen
  - Mandantenspezifische Ressourcen in UI-Elementen

UI-Erwartungsanalyse

Beim Bedienen einer interaktiven Softwareanwendung beobachtet der Anwender das an der Oberfläche wahrnehmbare Systemverhalten und trifft auf dieser Basis Annahmen über das Funktionieren der Anwendung.

Der Anwender trifft diese Annahmen auf Grund seiner Erfahrungen mit dem Bedienen von Maschinen und anderen Softwareanwendungen.

Um den Entwurf einer Oberfläche weiter zu präzisieren, kann der Entwickler bzw. der Analytiker zusätzliche Fragen über Erwartungen des Benutzers an das Bedienen der Software stellen.

Diese Fragen kann man wie folgt gruppieren.

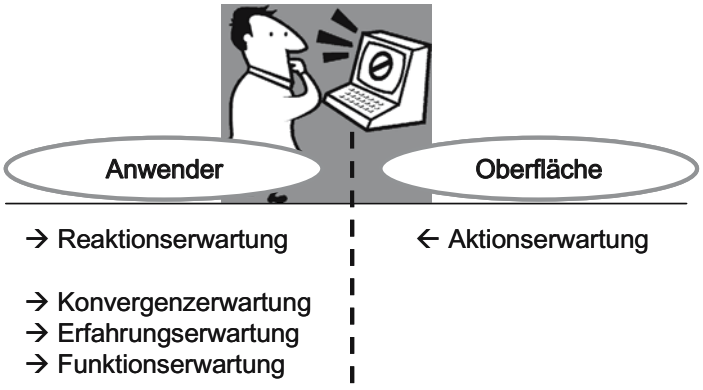


Bild 17: Erwartungen des Anwenders und der Oberfläche

Aktionserwartung

- Anwendersicht: Was muss ich (als Anwender) machen, damit die Anwendung etwas macht?
- Oberflächensicht: Oberfläche wartet auf Ereignisse, die sie interpretieren kann.
- Entwicklersicht: Über welche Eingabeelemente empfängt die Anwendung bzw. ihre Oberfläche Anweisungen des Benutzers?

Beispiel:

Um mit dem Erfassen eines neuen Kundendatensatzes zu beginnen, wird der Punkt ‚Neuer Kunde…' aus dem Hauptmenü gewählt.

Reaktionserwartung

- Anwendersicht: Was tut die Anwendung, wenn ich etwas mit ihr mache?
- Oberflächensicht: Alle Aktionen des Anwenders sollen durch eine wahrnehmbare Reaktion quittiert werden.
- Entwicklersicht: Welche Reaktionen zeigt die Anwendung auf Eingaben (oder auf Eingabeversuche) des Anwenders?

Beispiel:

Wenn ein Punkt aus dem Hauptmenü gewählt wird, wird die zu diesem Punkt passende Dialogseite eingeblendet.

Konvergenzerwartung

- Anwendersicht: Wenn ich das Gleiche wieder mache, macht die Anwendung dann auch wieder das Gleiche oder etwas anderes?
- Oberflächensicht: Die Reaktionen der Oberfläche sollen für den Anwender einem einfachen und leicht nachvollziehbaren Muster folgen.
- Entwicklersicht: Folgen auf die gleichen Eingaben (oder Eingabesequenzen) die gleichen Reaktionen der Anwendung? Wann reagiert die Anwendung anders?

Beispiel:

Wenn zum zweiten Mal hintereinander ‚Speichern' gewählt wird, könnte in der Statusleiste die Meldung ‚Seit dem letzten Speichern ist nichts geändert worden - Speichern nicht nötig.' erscheinen. Die Oberfläche könnte auch die Aufforderung zum Speichern ohne Rückmeldung ignorieren oder jedes Mal Speichern.

Erfahrungs- und Funktionserwartung

- Anwendersicht: Tun die Bedienungselemente das, wonach sie für den Anwender aussehen?
- Oberflächensicht: Die Bedienungsmetaphern sollen sich an der Alltagswelt orientieren.
- Entwicklersicht: Kann der Anwender dem Erscheinungsbild der Bedienungselemente und aus Erfahrungen mit

Anwendungen, die ähnlichem Zweck dienen, auf die Reaktion dieser Anwendung schließen?

Beispiel:

Anwender greifen intuitiv auf Erfahrungen und Metaphern zurück.

Wenn eine Schaltfläche mit „***Start***“ beschriftet ist, erwartet der unerfahrene Anwender eventuell, dass das gerade zuvor angeklickte Desktopsymbol gestartet wird.

Der erfahrene Anwender weiß, dass beim Klicken auf diese Schaltfläche ein Menü erscheint, in dem z.B. auch der Punkt „Computer ausschalten…“ enthalten ist, welchen er zunächst hinter dem Wort „Start“ nicht vermuten wird.

Möglicherweise wäre „Menü“ oder „Mach mal…“ eine erwartungskonformere Beschriftung der bewussten Start-Schaltfläche.

2.3 User Interface-Arten

Man kann Softwareoberflächen zum Beispiel nach Dialogform, Dialogmitteln (Modalität) oder nach Bedienungsparadigma (horizontal, vertikal) klassifizieren.

Untenstehend finden Sie die aus meiner Sicht wichtigsten Klassifizierungsmerkmale.

Dialogformen

- ***Directed Dialog*** (angeleiteter Dialog): Computerinitiierte Dialoge, z.B. grafische oder gesprochene Auswahlmenüs.
- ***Form Filling Dialog***: Eine Form des angeleiteten Dialogs, bei der vor allem die vom Computer angebotenen Formulare (z.B. eine Hotelbuchung) ausgefüllt werden sollen. Der Workflow der Anwendung besteht aus mehreren aneinander gereihten Formularen.
- ***Systeminitiierter Dialog***: Ebenfalls eine Form des angeleiteten Dialogs. Der Computer informiert den Anwender über ein Ereignis und bietet in der Regel Reaktionsmöglichkeiten an, z.B. in Form eines Auswahlmenüs.
- ***Offener Dialog*** (nicht angeleiteter Dialog): Der Mensch erklärt, was er möchte; der Computer grenzt die Antwortmöglichkeiten ein.

Die meisten Anwendungen, vor allem Geschäftsanwendungen, sind Form Filling Dialog-Anwendungen. In der Regel enthalten alltagsübliche Anwendungen auch systeminitiierte Dialoge. Der offene Dialog zwischen Software und Anwender ist bis auf einzelne gesprochene Steuerbefehle in spezialisierten Anwendun-

gen (z.B. Maschinensteuerung, Radionavigationssysteme) noch nicht weit verbreitet.

Dialogmodalitäten

Dialogmodalität bezeichnet die Mittel, die die Oberfläche (überwiegend) dem Anwender zur Dialogführung bereitstellt.

Befehlszeilen

Command Line Interpreter (CLI):

- Der Anwender gibt Befehle und Befehlsparameter in einer Kommandozeile an.
- Der Befehl wird mit einer Freigabetaste (Enter) abgeschlossen und an den Kommandointerpreter übergeben.
- Nach dem Ausführen des Kommandos ist der CLI eingabebereit für den nächsten Befehl.
- Beispiel: Unix Command Shell.

Textmasken

Text User Interface (TUI):

- Die Benutzerschnittstelle ist zeichen- bzw. textbasiert (üblicherweise 25 Zeilen á 80 Zeichen).
- Aus Eingabefeldern, beschreibenden Texten und grafischen Zeichen werden auf TUIs „Ein- und Ausgabe-Masken“ zusammengesetzt.
- Die Eingabe von Daten und Befehlen erfolgt über diese Masken durch Tastatureingaben. Üblicherweise werden textbasierte Oberflächen auch über Tastenkombinationen und Menüs bedient.
- Prominente Vertreter: Word Star, Norton Commander, Lotus 123

Grafische Kontrollelemente

Graphical User Interface (GUI):

- Die Benutzerschnittstelle ist grafikbasiert.
- Verschiedene Bedienungselemente wie Fenster, Schaltknöpfe, Regler und Eingabefelder werden mit der Maus, Tastatur und eventuelle weitere Eingabegeräte (z.B. Grafiktablett) bedient.
- Die meisten kommerziellen Anwendungen werden heute mit einem GUI ausgestattet.
- Typische Vertreter sind Windowsanwendungen für den täglichen Büroeinsatz (OpenOffice oder MS Office), Apple Macintosh-Anwendungen oder GIMP (GNU Image Mani-

pulation Program, Bildbearbeitungssoftware), z.B. unter KDE oder GNOME.

Voice User Interface (VUI): *Sprachsteuerung*

- Der Anwender steuert die Anwendung mit gesprochenen Befehlen.
- Die Ausgaben des UI bestehen entweder aus vorab aufgezeichnetem Ton oder aus über eine synthetische Stimme ausgegebenem Text.
- Die Eingaben erfordern eine Spracherkennung. Nach der Spracherkennung das Interpretieren und Ausführen der Anwenderanweisung Ähnlichkeit mit der Arbeitsweise eines Command Line-Interpreters.
- Oft werden VUIs mit anderen IO-Modalitäten, wie Zeicheneingabe über Tastatur (z.B. DTMF Signale an Telefonen) oder mit GUIs (z.B. in Fahrerassistenzsystemen) kombiniert.

Haptische User Interfaces (HUI): *Steuerung durch Körperbewegung*

- HUIs reagieren auf Bewegungen des Benutzers, z.B. durch Datenhandschuh oder durch Kamerabeobachtung.
- Sie können auch mit haptischen Rückkopplungen kombiniert sein.
- Beispiele für haptische User Interfaces sind Virtual Reality oder Augmented Reality.
- Ein repräsentatives Beispiel für ein UI mit haptischer Rückkopplung sind Flugsimulatoren, bei denen z.B. Beschleunigung und Bremsen für den Anwender körperlich spürbar sind.

Horizontale und vertikale Oberflächen

Man kann in einer Anwendungsoberfläche zwischen einer ***horizontalen (ablauforientierten)*** und einer ***vertikalen (bearbeitungsgegenstandorientierten)*** Bedienungsform unterscheiden.

Oft werden in einer Softwareanwendung beide Bedienungsformen nebeneinander und miteinander vermischt angewandt. Je nachdem, welche Bedienungsform überwiegt, kann man die gesamte Anwendungsoberfläche als vertikal oder horizontal bezeichnen.

Vertikales User Interface

Vertikale User Interfaces sind solche, in denen Objekte direkt manipuliert werden. Über diese Manipulation löst der Anwender komplexe Funktionen wie z.B. Grafik- und Textverarbeitung oder 3D-Modellierung (CAD) aus. Diese Anwendungen haben eine ***vertikale*** UI-Ausdehnung, die dadurch gekennzeichnet ist, dass die bearbeiteten Objekte selbst als (relativ wenige) Kontrollelemente für vielfältige Umformungen dienen.

Horizontales User Interface

Horizontale User Interfaces treten hingegen vor allem in wirtschaftlich orientierten Anwendungen auf, wo der Fokus auf Workflows und auf intensiver Ein- und Ausgabe von Daten liegt. Diese Anwendungen haben eine ***horizontale*** UI-Ausdehnung, die durch die Aneinanderreihung von vielen elektronischen Formularen gekennzeichnet ist.

Vertikale User Interfaces haben im Vergleich zu horizontalen User Interfaces weniger Dialogseiten und Kontrollelemente. Ihre Kontrollelemente sind aber komplexer, weil sie mit der grafischen Repräsentation der Anwendungsdaten, z.B. einem CAD-Modell oder dem bearbeiteten Text, verzahnt sind.

2.4 Funktionsweise einer Softwareoberfläche

Eine Softwareoberfläche umfasst diejenigen Teile der Software, die in Form von Ausgaben, Eingabeelementen oder Dialogführung zwischen dem Anwender und der Funktionalität der Anwendung vermitteln.

Die Softwareoberfläche ist ein Vermittler zwischen den Dialogpartnern Anwender und Softwarefunktionalität.

Die Oberfläche ist der dem Anwender (Mensch) zugewandte Teil der Software (Maschine) und zuständig für den Austausch von Informationen zwischen den Anwendungsfunktionen (funktionale Maschine) und der Anwendungsumwelt (im betrachteten Fall der Anwender).

Die für den ***Anwender (Mensch)*** bestimmten Informationen müssen für diesen wahrnehmbar und interpretierbar sein.

Die für die ***funktionale Anwendung (Maschine)*** bestimmten Informationen müssen für diese verarbeitbar und eindeutig sein.

Es handelt sich bei einer Anwendungsoberfläche also um eine Art ***Übersetzer und Vermittler***, dessen Aufgabe es ist, die Eingaben des Menschen an den Rezeptoren / Steuerungseinrichtungen / Kontrollelementen der Software an die Anwendung weiterzureichen.

In der umgekehrten Richtung wandelt die Oberfläche die Daten der Anwendung in mensch-wahrnehmbare Informationen d.h. in Ausgaben (i.d.R. in Form von visuellen und akustischen Äußerungen) um.

In beiden Richtungen findet dabei immer das Verhandeln der Oberfläche gegenüber der Funktionalität statt.

Bild 18: Empfangen, Interpretieren, Verarbeiten, Manifestieren

Eine Anwendungsoberfläche ist in diesem Kontext ein interaktives System aus

- Eingabeempfängern (Rezeptoren),
- Interpretations- und Verarbeitungsregeln (Prozessoren),
- Ausgabemechanismen (Manifestoren).

Oberfläche= Frontend, Anwendung= Backend

In kommerziellen Projekten bei Rechenzentren und Banken wird Oberfläche auch als ***Frontend*** und der funktionale Teil der Anwendung als ***Backend*** bezeichnet.

Entsprechend dem Zweck der über die Oberfläche zugänglichen Funktionalität manifestiert die Anwendungsoberfläche in geordneter Weise das ***Empfangen***, das ***Interpretieren*** und das ***Verarbeiten*** von Aktionen des Anwenders und von Signalen des funktionalen Systems unterhalb der Oberfläche.

In den Grundzügen ist dieses Schema der vermittlerischen Funktionsweise einer Softwareoberfläche analog zu den koordinierenden Aufgaben eines Betriebssystemkerns [Kidder 1982, S.140ff].

Experimentelle User Interfaces und die KI-Forschung (Künstliche Intelligenz) liefern für die Softwareoberflächen des täglichen Gebrauchs Ansätze für plastische Erklärungsmodelle [Rose 1985, S.60f]. So kann man zum Beispiel die Zusammenarbeit zwischen einer Oberfläche und der funktionalen Anwendung mit dem Zusammenspiel zwischen Sinnesorganen, Gehirn und Körperfunktionen vergleichen.

Interaktion und Mechanik

Software funktioniert nicht mechanisch

Interaktive Werkzeuge können in materielle und immaterielle Systeme gegliedert werden. Ein Buch ist materiell und unterliegt dadurch physisch-mechanischen Gesetzen. ***Software ist immateriell***: Sie unterliegt keinen mechanischen Gesetzmäßigkeiten. Dies ist eine ihrer Stärken und zugleich ein Risiko.

Die Interaktion eines Papierbuches mit seinem Nutzer ist mechanisch und kulturell klar geregelt. Ausnahmen bestätigen die Regel. Wenn beim Blättern in einem Buch plötzlich die eben umgeblätterte Seite Flammen fängt oder Buchstaben sich in Nichts auflösen, wenn das Buch zu Staub zerfällt oder andere okkulte Aktivitäten entwickelt, wissen wir, dass dies nach physikalischen Gesetzen nicht normal ist.

Bei dem Versuch, das Verhalten von Software zu verstehen, greift der Nutzer auf seine Erfahrungen mit der realen Umwelt zurück. Er erwartet intuitiv, dass die Software gewissermaßen den Gesetzen der Mechanik folgt, was aber oft nicht der Fall ist.

Bild 19: Software kann man nicht mechanisch wahrnehmen

Wenn eine Software nicht die intuitiv erwarteten Aktivitäten unternimmt, z.B. wenn Dialogseiten plötzlich nicht an gewohnter Stelle erreichbar sind, oder wenn Dialogelemente plötzlich erscheinen oder verschwinden, sprechen wir von Features oder von Fehlern und legen ggf. im Einzelfall das Sollverhalten fest.

Kognitive Wissenschaften und die Usability-Forscher beschäftigen sich unter anderem damit, wie die Informationen an der Oberfläche dargestellt werden und wie Eingabeelemente aufgebaut sein müssen, damit sie für den Anwender intuitiv und optimal verwendbar sind. Während die mechanischen Metaphern (z.B. Einrasten einer Schaltfläche beim Draufklicken) von allen Menschen ähnlich wahrgenommen werden, hängen andere Oberflächeneigenschaften (z.B. Farben, Piktogramme) vom kulturellen und sozialen Hintergrund der Anwendergruppe ab.

Zum Beispiel haben ein 11-jähriges Mädchen aus Tokyo und ein 60-jähriger griechischer Weinbauer ganz unterschiedliche Wahrnehmungen von einer und derselben Anwendungsoberfläche (falls der Weinbauer überredet werden kann, sich eine anzusehen).

Eine konsequente Anlehnung der Bedienung und des Verhaltens von Oberflächenelementen an mechanische Gesetze könnte sich positiv auf die intuitive Anwendbarkeit von Softwareprogrammen auswirken. Hier können UI-Entwickler und selbst Usability-Forscher noch von Weinbauern lernen.

Verlauf des Anwender-Anwendung-Dialogs

Das folgende Funktionsmodell erklärt die Vorgänge, die in der Software stattfinden, wenn ein Anwender und eine Anwendungsoberfläche in einen Dialog treten, das heißt also wenn der Anwender Bedienungselemente der Anwendungsoberfläche betätigt, um eine von ihm gewünschte Funktion der Anwendung auszuführen.

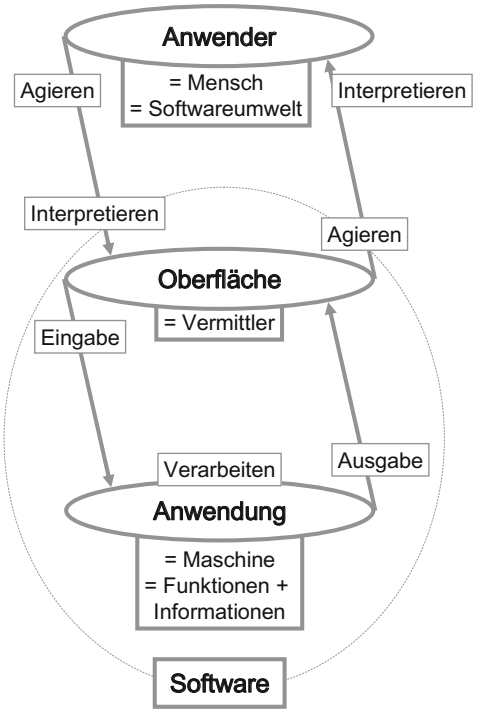


Bild 20: Dialog zwischen Anwender und Oberfläche

An dem oben skizzierten Dialog beteiligt sind die Akteure:

- Der Anwender,
- Die Oberfläche im Sinne des Vermittlers zwischen Anwender und Anwendung,
- Die Anwendung im Sinne der durch die Oberfläche erreichbaren Funktionalitäten und Informationen.

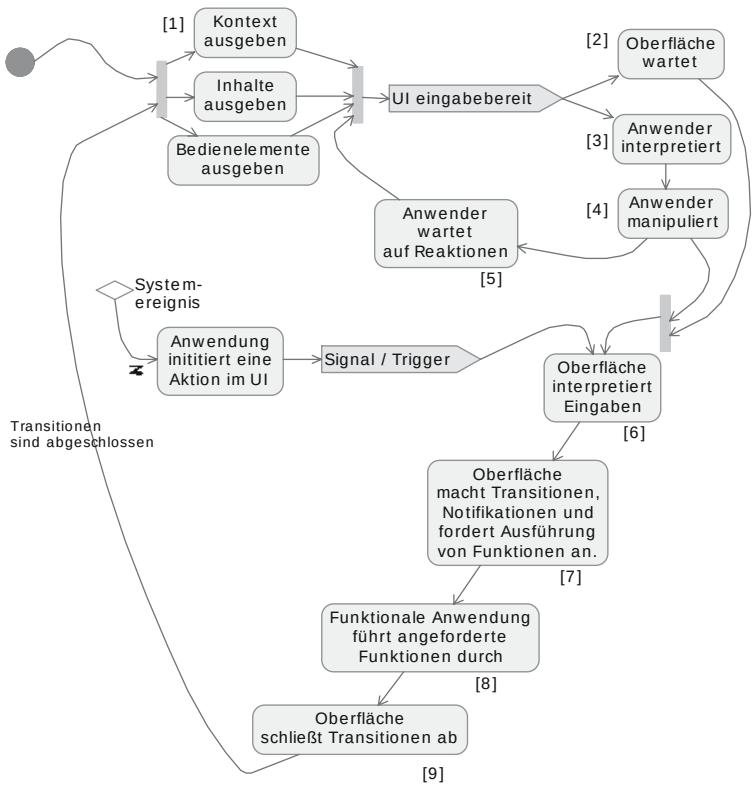
Der Dialog zwischen Anwender und Anwendung findet ***indirekt*** über die Vermittlung durch die Oberfläche statt.

Der Anwender ist die Umwelt der Anwendung. Dieser Akteur lässt sich zu „Mensch oder Maschine“ erweitern. Mit dieser Erweiterung könnte man den Lösungsraum der Schnittstellenkommunikation im Allgemeinen, zum Beispiel den Dialog zwischen zwei Anwendungsoberflächen, untersuchen.

Entsprechende Lösungen verfolgt die SOA (Service-orientierte Architektur), indem in diesem Architekturansatz die Funktionalität der Anwendung als Prozessbausteine in Form von Diensten bereitgestellt wird. Die User Interface-orientierte Architektur beschränkt sich auf die Untersuchung des Dialogs des Anwenders mit der Oberfläche und der Oberfläche mit der Funktionalität der Anwendung.

Um ein Ablaufschema für diesen Mensch-Maschine-Dialog aufzustellen, kann man die Dialogsituation wie oben skizziert aus dem Blickwinkel der Softwareoberfläche analysieren.

Bild 21: Mensch-Maschine-Dialog aus der Sicht des UI



Ein Zyklus des in der Grafik skizzierten Dialogs zwischen Anwender, Oberfläche und Anwendung hat folgende Schritte:

1. ***Oberfläche*** gibt Inhalt und Kontextinformationen aus und stellt Eingabemöglichkeiten zur Verfügung. Das User Interface zeigt dazu z.B. eine Dialogseite an und / oder spielt eine Sprachausgabe ab.
2. ***Oberfläche*** wartet, was nun passiert. Die Anwendung wartet i.d.R. nicht mit der Oberfläche, sondern erwartet ihrerseits Systemereignisse und / oder Signale von der Oberfläche.

3. ***Anwender*** interpretiert die Ausgaben. Der Anwender sieht bzw. hört, was die Oberfläche ausgibt und macht sich einen Reim drauf.
4. ***Anwender*** manipuliert Bedienungselemente (ob dies aus Sicht der Oberfläche „echte Eingaben" sind, ist zu diesem Zeitpunkt noch nicht entschieden).
5. ***Anwender*** wartet, was nun passiert.
6. ***Oberfläche*** interpretiert die Eingaben im Kontext der Anwendung (Die Oberfläche befragt dazu ggf. den Datenhaushalt der Anwendung).
7. ***Oberfläche*** reagiert (anwendergetriggert) bzw. agiert (anwendungsgetriggert) z.B. durch
   - Transitionen (Zustandsübergänge in der Oberfläche,
   - Neuberechnung von Guards (Bedingungen, die an das Durchführen von Transitionen geknüpft sein können) unter Verwendung des Datenhaushalts der Anwendung,
   - Oberflächen-Dienste (z.B. Rückmeldung an Anwender), oder durch
   - Anfordern des Ausführens von Anwendungsfunktionalitäten.
8. ***Anwendung*** führt (aus Sicht der Oberfläche synchron oder asynchron, intern in der Regel asynchron) die angeforderten Funktionalitäten aus bzw. liefert die abgefragten Daten.
9. ***Oberfläche*** schließt die Reaktionen ab und läutet die Bereitstellung der aktuellen Ausgaben, Kontextinformationen und Bedienelemente ein.
10. Weiter mit Schritt 1.

Checkliste für das Prüfen von Beschreibungen

Anhand dieser Dialogschritte kann z.B. die Vollständigkeit der Beschreibung eines Dialogs in der Benutzeranleitung bewertet werden.

Unterschied zwischen Aktion-Interpretation-Reaktion und Eingabe-Verarbeitung-Ausgabe

Wesentlich für das Verstehen des Mensch-Maschine-Dialogs ist es, zwischen dem kommunikationsorientierten Schema ***Aktion-Reaktion*** und dem funktionsbezogenen Schema ***Eingabe-Ausgabe*** zu unterscheiden.

Auf der Ebene von Eingabe-Ausgabe hat das Interpretieren von Benutzeraktionen bereits vorher stattgefunden. In der Anwendung wird daraufhin eine definierte Funktion ausgelöst, die ein definiertes Ergebnis liefert.

Auf der Ebene von Aktion-Reaktion findet hingegen ein gegenseitiges Interpretieren statt. Der Anwender trifft Annahmen über die von der Oberfläche bereitgestellten Informationen, Bedie-

nungselemente und dahinter stehende Funktionen. Die Oberfläche trifft Annahmen über das Wahrnehmen und Agieren des Anwenders und über die verfügbaren Funktionen und Daten der funktionalen Anwendung.

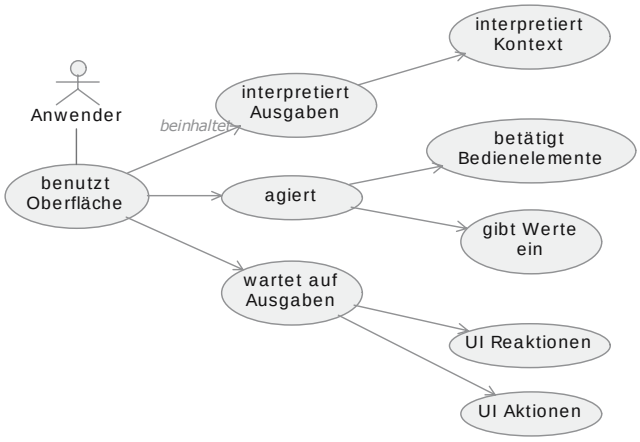
Dialog bzw. Kommunikation mit dem Anwender (im Sinne von Expression und Signalisation) ist Teil des Verhaltens einer Softwareoberfläche. Auf Seite des Anwenders und auf Seite der Oberfläche besteht die eigentliche Kommunikation aus Kodierungen und Entkodierungen in einem gemeinsamen Begriffsraum statt [Ebeling 1988, S.97-99]. Dieser Begriffsraum wird durch die Ein- und Ausgabeelemente der Softwareoberfläche bereitgestellt.

Use Cases beim Bedienen des UI

Die Verwendungsfälle (Use Cases) beim Bedienen einer Softwareoberfläche ***aus Sicht des Anwenders*** sind:

- Die Oberfläche (Ausgaben der Anwendung im Kontext der Präsentation) interpretieren,
- Eingabe und Steuerungselemente betätigen,
- Rückmeldung über (laufende) Verarbeitung der Eingaben erhalten.

Bild 22: Use Cases - Anwendersicht



Die Verwendungsfälle ***aus Sicht der Oberfläche*** sind:

- Aus- und Eingabeelemente (für den aktuellen logischen Anwendungsschritt) bereitstellen,
- Eingaben des Anwenders interpretieren,
- Signale der Anwendung interpretieren,
- Datenkontext der Anwendung abholen und auswerten,

- Den Zustandsvektor der Oberfläche ermitteln (Bedingungen für Aktionen bzw. Situationen berechnen),
- Oberflächen-Transitionen (Übergänge in der Oberfläche) auslösen,
- Ausführung von Anwendungsfunktionen auslösen.

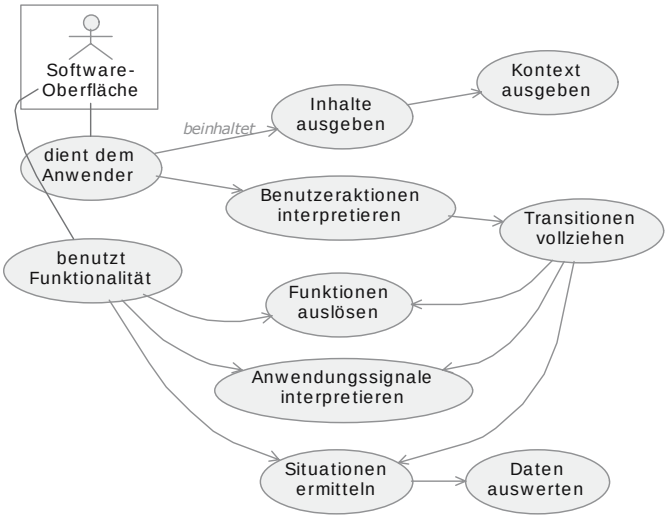


Bild 23: Use Cases - Sicht der Oberfläche

Die Verwendungsfälle ***aus Sicht der funktionalen Anwendung*** sind:

- Requests der Oberfläche über Ausführung Funktionalitäten quittieren und umsetzen,
- Requests der Oberfläche über Bereitstellung des Datenkontexts quittieren und umsetzen,
- Signale an die Oberfläche senden (Die Anwendung sendet namenlose Signale (die unterschiedliche Priorität haben können), um der Oberfläche zu signalisieren, dass diese agieren soll).

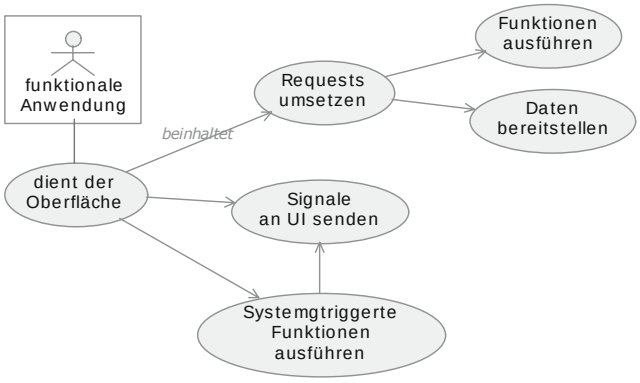


Bild 24: Use Cases - Sicht der funktionalen Anwendung

2.5 Informationsgehalt des UI

Der Inhalt bzw. Informationsgehalt einer Softwareoberfläche (bzw. einer Oberflächenbeschreibung) besteht, vereinfacht gesagt, aus Antworten auf Fragen, die gestellt werden müssen, wenn man als Entwickler eine Oberfläche spezifizieren bzw. konstruieren soll.

Anhand der Leitfragen zur UI-Beschreibung (siehe oben) wird schnell ersichtlich, dass die sichtbaren Kontroll- und Darstellungselemente nur einen Teil der Oberfläche ausmachen.

Der weitaus größere Teil der Fragen hat mit dem Verhalten der Oberfläche, den Abläufen aus Anwendersicht und dem Zusammenspiel der Bedienungselemente zu tun.

Es ist an der Zeit, die beiden schon benutzten Begriffe - Kontrollelement bzw. Bedienelement und Verhalten - zu definieren.

Kontrollelement, Bedienelement

Kontrollelement (bzw. Bedienelement) ist ein Ding, das Eingaben des Anwenders entgegennimmt und / oder Ausgaben der Software präsentiert.

Es gibt eine Reihe spezialisierter Bedienelemente, z.B. Druckknöpfe, Eingabefelder, Ausgabefelder, Regler, Auswahlfelder, Fortschrittanzeiger und so weiter. Weiter in diesem Kapitel finden Sie eine detaillierte Klassifikation der mir bekannten Kontrollelemente.

Verhalten

Verhalten ist die Art und Weise, wie die Anwendung agiert und reagiert. Das Verhalten einer Softwareoberfläche ist durch die Summe ihrer Aktionen und Reaktionen festgelegt.

Interaktion

Als Oberbegriff für Aktionen und Reaktionen mit den dazugehörigen Verknüpfungen verwende ich den Begriff Interaktion.

Interaktionen verknüpfen die für den Anwender wahrnehmbaren Teile der Oberfläche (also in der Regel Kontrollelemente) mit den Funktionen der Anwendung und die Kontrollelemente untereinander.

Interaktionen werden durch Ereignisse ausgelöst. Ereignisse können durch Aktionen des Anwenders oder durch Initiative der Anwendung (z.B. in Folge eines Systemereignisses) ausgelöst werden. Die Auswertung der Bedingungen, wie die Oberfläche fallabhängig auf Ereignisse reagiert, gehört ebenfalls zu den Interaktionen.

Das Verhalten der Oberfläche setzt sich also aus Ereignissen, Reaktionsbedingungen und Reaktionen zusammen. Eine Interaktion ist ein strukturiertes Gebilde aus

- Ereignis mit Objektbezug (d.h. der Bereich der Oberfläche, auf den sich das Ereignis bezieht, z.B. ein einzelnes Kontrollelement),

- 0..* Bedingungen, die zutreffen müssen, um eine oder mehrere bestimmte Reaktionen auf das Ereignis auszulösen,
- 0..* Reaktionen, die beim Zutreffen der Bedingungen folgen.

Natürlich kann ein und das selbe Ereignis bei gleichem Objektbezug unterschiedliche Reaktionen auslösen. Jeder Satz von 0..* Bedingungen, der zu einem anderen Satz von 0..* Reaktionen führt, zählt als eine eigene Interaktion.

Wie leicht zu erkennen ist, ist die Beschreibung des Verhaltens einer Softwareoberfläche viel aufwändiger, als die Festlegung und Anordnung der Kontrollelemente auf dem Bildschirm.

Nach Raskin sollte eine Softwareoberfläche mit dem Anwender sogar auf Basis der kleinsten bedeutungsvollen Eingabeeinheit (also einzelne Zeichen und Mausbewegungen) interagieren [Raskin 2001, S.156].

Bei Voice User Interfaces sind die Kontrollelemente nur durch Ereignisse und Interaktionen (Sprechen, Hören) wahrnehmbar. Bei Graphical User Interfaces (GUI) beschränken sich die meisten Werkzeuge auf die Unterstützung des Festlegens von Dialogseiten (also Kontrollelemente und deren Anordnung). Sie überlassen die Definition des Verhaltens der Oberfläche der Programmierung in einer Programmiersprache. Das führt zu Brüchen in der Oberflächendefinition (die Oberfläche ist teils im GUI-Werkzeug beschrieben und teils programmiert) und zu Technologieabhängigkeit (man hat sich auf die gewählte Programmiersprache festgenagelt).

Einige GUI-Werkzeuge versuchen, das Verhalten der Anwendungsoberfläche mit State Charts oder mit Entscheidungstabellen abzubilden. Wie wir noch sehen werden, genügt das nicht für eine effektive Verhaltensbeschreibung einer Softwareoberfläche, weil der Objektbezug und die Reaktionsbedingungen sehr komplex werden können.

Ausgerüstet mit der Definition von Kontrollelement, Verhalten und Interaktion können wir zur weiteren Untersuchung der Bestandteile der Softwareoberfläche schreiten.

In einem User Interface lassen sich leicht zwei voneinander verschiedene Sorten von Bestandteilen identifizieren:

- ***Ein- und Ausgabeelemente***: Das sind die vom Anwender wahrnehmbaren Informationsträger.
- ***User Interface-Logik***: Das sind die logischen Konstrukte, die das Zusammenspiel der Ein- und Ausgabeelemente organisieren.

2.6 Kontrollelemente

Unter Kontrollelementen werden im Folgenden sowohl Eingabe- als auch Ausgabeelemente verstanden.

Kontrollelemente versetzen den Anwender in die Lage, die Ergebnisse der Anwendungsoperationen wahrzunehmen und seinerseits Aktionen in der Anwendungsoberfläche anzustoßen und die hierfür notwendigen Informationen (Eingabedaten) einzugeben.

Widgets sind eine Weiterentwicklung von Kontrollelementen. Ein Widget entspricht der Zusammenfassung von Layout- und Verhaltensinformationen in einem Objekt.

Das Aussehen eines Widgets wie eines Kontrollements hängt oft von seinem Zustand ab. Der Zustand wird zur Laufzeit durch Auswertung von Funktionsergebnissen und Statusinformationen abgefragt. Zum Beispiel wird eine Schaltfläche oft grau dargestellt, wenn die durch die Schaltfläche aktivierbare Funktionalität zu diesem Zeitpunkt nicht zur Verfügung steht.

Unabhängig von ihrem Erscheinungsbild, das sehr unterschiedlich sein kann, findet man in den heute üblichen Softwareanwendungen folgende Klassen von (logischen) Kontrollelementen.

Schaltflächen

- Pushbutton
- Toggle Button

Auswahlfelder

- Drop Down-Liste
- Menüleiste
- Kontextmenü / Einzelmenü
- List Box
- Slider / Schieberegler
- Spincontrol / Drehregler
- Radio-Auswahlfeld-Gruppe
- Optionsschaltfläche / Ankreuzfeld / Checkbox

Text- und Zahlen Ein- und Ausgabe

- Formatiertes Texteingabefeld/Eingabeschablone
- Combobox
- Mehrzeilige Texteingabe

Notifikationsfelder

- Gauge / Fortschrittsanzeiger
- Statusanzeiger

Strukturierungselemente

- Groupbox / Dialogabschnitt
- Positionierungsraster
- Statusleisten

Akustische Ein- und Ausgabe

- Sounds
- Sprachmenüs
- Spracheingabeempfänger

Gestaltungselemente

- Geometrische Formen (z.B. Linien)
- Pixelgrafiken
- Vektorgrafiken
- Piktogramme

Bewegte Bilder

- Videosequenzen
- Animationen

Eine detaillierte Beschreibung der Funktion und des Einsatzzwecks der einzelnen Kontrollelemente von GUIs finden Sie z.B. in [Wessel 2002] oder in [Shneiderm. 2002].

Logische und physische Kontrollelemente

Logische Bedienelemente

Logische Bedienelemente sind aufzählbar, das heißt es gibt eine begrenzte Anzahl davon. Die obige Aufzählung zeigt die in heutigen Oberflächen üblichen logischen Kontrollelemente. Ich bezeichne sie als logische Elemente, weil sie zwar eine bestimmte Grundform haben, aber ihre grafische Ausgestaltung und mechanische Umsetzung nicht festgelegt sind.

Zum Beispiel ist für das logische Funktionieren einer Checkbox nicht von Bedeutung, ob sie durch einen umzulegenden Hebel oder durch ein Ankreuzfeld repräsentiert wird.

Physische Bedienelemente

Physische Bedienelemente sind Rendering-behaftete Inkarnationen von logischen Kontrollelementen. Sie können, wenn man von ihrem Aussehen absieht, auf logische Bedienelemente zurückgeführt werden.

Weitere Fragestellungen zu logischen Bedienelementen können sein:

- Wo liegt die Grenze zwischen logischen Controls und realen Controls?
- Welche Informationen werden an der Schnittstelle zwischen logischen Controls und realen Controls ausgetauscht?
- Welche Events müssen die realen Controls an logische Controls weitergeben bzw. senden?
- Welche Events sind für welchen logischen Control definiert?
- Welche Events werden an der logischen Schicht eines Kontrollelements (am Beispiel eines komplexen Kontrollelements wie GridControl oder DataTable) benötigt?
- Wie hängt ein logisches Bedienelement vom Anzeige- und Bedienungskonzept (Mouse based GUI, Touch Screen, Sprachbefehle, Mischmodale Eingaben) ab?

2.7 User Interface-Logik

Das hier als User Interface-Logik bezeichnete Geflecht hinter den Kontrollelementen ist für den Anwender schwieriger zu erfassen als die Kontrollelemente. Diese Schwierigkeit ist leicht nachvollziehbar, weil die User Interface-Logik für den Anwender nur indirekt durch das Verhalten der Kontrollelemente wahrnehmbar ist.

Typischerweise unterscheiden die Anwender nur zwischen der „***Bildschirmmaske***“ und dem „***Programm hinter der Maske***“, ohne weiter zwischen ***Präsentationslogik***, ***Schnittstellenlogik*** oder ***Geschäftslogik*** zu differenzieren.

Wenn es aber darum geht, eine Softwareoberfläche zu konstruieren, ist eben dieser unsichtbare Teil der Oberfläche, das Programm hinter der Maske, das zwischen den Kontrollelementen und den Anwendungsfunktionen vermittelt, interessant.

Für diese Innenansicht der Softwareoberfläche kann man das strukturierte Geflecht der User Interface-Logik in die Bestandteile

- „***Ablaufnetz***“,
- „***Dialogseiten***“ bzw. allgemeine „Dialogeinheiten“ und
- „***Kontext***“

gliedern.

Dieser Gliederungsvorschlag fußt auf der eigenen Erfahrung, dass unabhängig von der Architektur einer Software die Entwickler stets zwischen

- „Kontrollfluss“ (Controller),
- „Dialogen“ (Views mit Controllern) und
- „Daten“ (Model)

unterscheiden (vergleiche hierzu auch MVC-Entwurfsmuster).

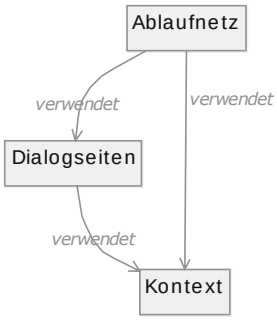


Bild 25: Ablaufnetz, Dialogseiten und Kontext

Das ***Ablaufnetz*** steuert den Ablauf der Softwareoberfläche und verwendet dabei die ***Dialogseiten*** und den ***Kontext***.

Ablaufnetz

UI-Logik, die den Ablauf der Anwendung oberhalb der Kontrollelemente und einzelner Dialogseiten steuert:

- Hierarchie der logischen Schritte der Anwendung
- Klassifizierung der logischen Schritte in Sequenzschritte und Services (Popups, Notifikationen)
- Interaktionen beim Betreten und Verlassen der logischen Schritte
- Sequentielle und nichtsequentielle Übergänge zwischen logischen Schritten
- Ablaufkontrolle und Navigationsinstrumente

Dialogseiten

UI-Logik, die innerhalb einer Dialogeinheit stattfindet:

- Dialogschnittstelle für einen logischen Schritt
- Anordnung der Kontrollelemente auf dem Bildschirm
- Interaktionen für die Kontrollelemente

Kontext

UI-Logik, die den Zusammenhang zwischen dem Zustand der Oberfläche und dem Zustand der funktionalen Anwendung herstellt:

- Verwendung von Daten
- Verwendung von Funktionen
- Prozesslogik
- Statusermittlung durch Situationsindikatoren
- Verwendung von Widgets
- Verwendung von Layouts
- Verwendung von Datenmustern
- Bezugnahme auf Anforderungen

2.8 Nutzen

Die Erforschung der Bestandteile und Zusammenhänge hat ein differenziertes Bild der Oberfläche geliefert. Eine Softwareoberfläche ist demzufolge keine lose Ansammlung von beliebigen Bedienungselementen, sondern ein strukturiertes Gebilde mit einer klaren Gliederung.

Diese Gliederung ermöglicht:

- Vollständigkeitsbewertung von UI-Beschreibungen,
- Vergleich von UI-Beschreibungen,
- Strukturiertes Aufnehmen und Dokumentieren von User Interface Anforderungen,
- Aufstellen von funktionalen Anforderungen an Werkzeuge zur UI-Modellierung,
- Bewerten von Verfahren zur UI-Modellierung.

Die nächsten Kapitel schöpfen den Nutzen dieser User Interface-Gliederung aus.

Kapitel 3: Werkzeuge

… in dem Anforderungen an UI-Werkzeuge sowie unterschiedliche Werkzeugtypen und konkrete Werkzeuge vorgestellt werden.

Kapitelziele

- *Wissen, welche typischen Phänomene beim Entwickeln eines User Interface auftreten.*
- *Wissen, nach welchen Kriterien Werkzeuge zur UI-Entwicklung bewertet und ausgewählt werden können.*
- *Überblick über Werkzeuge zur UI-Entwicklung.*

Woraus und womit wird ein User Interface gebaut?

Wie bei jedem Konstruktionsprojekt, so auch bei Software (und damit auch bei ihrer Oberfläche), taucht bereits zu Beginn des Projekts neben der Frage „Was wird gebaut?“ bald die Frage „Woraus und womit wird gebaut?“ auf.

In einem Softwareprojekt können konkrete Fragen der Entwickler an den Architekten zum Beispiel so lauten:

- Mit ***welchem Werkzeug*** soll die Softwareoberfläche entwickelt werden?
- ***Wie gut*** ist das zur Auswahl stehende Werkzeug zum Bauen der Oberfläche geeignet?
- Nach welchen ***Kriterien*** wurde die Werkzeugauswahl getroffen?

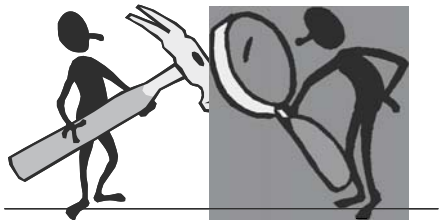


Bild 26: Wissen, worauf es beim UI-Werkzeug ankommt

Um diese Fragen fundiert zu diskutieren, sollten Architekt und Entwickler neben der Kenntnis der technischen und organisatorischen Rahmenbedingungen des Projekts wissen, wie man ein Werkzeug zur User Interface-Entwicklung auswählt.

Zu diesem Wissen gehören:

- Kenntnis der typischen ***Phänomene*** beim Festlegen, Detaillieren und Umsetzen der Eigenschaften einer Oberfläche.

- Kenntnis der ***Anforderungen*** an Features einer UI-Entwicklungsumgebung.
- Überblick darüber, wie man die Features einer UI-Entwicklungsumgebung einschätzt und wie man ***bewerten*** kann, wie gut sie die Anforderungen erfüllen.
- Überblick über die Werkzeuge, die zur ***Auswahl*** stehen.

3.1 Typische Phänomene des UI-Entwickelns

In typischen kommerziellen Softwareentwicklungsprojekten werden in allen Projektphasen Dialogseiten entworfen und verworfen, angepasst und geändert. Die Projektbeteiligten diskutieren laufend den Dialogfluss. Sie detaillieren die Reihenfolge der Bildschirmseiten und passen das Verhalten einzelner Kontrollelemente an.

Herausforderung des UI-Entwickelns: Überblick behalten

Beim Entwickeln von User Interfaces gilt es, den Überblick zu behalten. Man muss eine Menge Anforderungen, technische Rahmenbedingungen und Ergebnisse vieler Workshops und Kantinendiskussionen in ein stimmiges Gesamtbild bringen und das Bild laufend aktuell halten.

Dabei treten gewöhnlich Fragestellungen der Art „Wie funktioniert alles zusammen?“, „Wie spezifiziere ich das?“ oder „Wie soll ich das programmieren?“ auf.

*n * (n - 1) / 2 mögliche Beziehungen zwischen n UI-Elementen*

Bei einer Anwendung der nichttrivialen Größe (so ab 50 Dialogseiten) häufen sich die Fragen nach Formulierbarkeit und Beherrschbarkeit der User Interface Definition (ob als Code oder als Spezifikation). Die Komplexität einer User Interface Definition wächst schnell, weil die Anzahl der möglichen Beziehungen (und damit die Anzahl der gegebenenfalls zu beschreibenden Beziehungsregeln) zwischen n UI-Elementen mit dem Faktor n * (n - 1) / 2 quadratisch wächst. Die Formel gibt Anzahl der möglichen direkten Kommunikationswege zwischen n Partnern bzw. interaktiven Elementen an.

Typische Stolpersteine

Die auftretenden Fragen betreffen in erster Linie

- das Formulieren der Ablauflogik,
- das Wiederverwenden von Festlegungen,
- das Bezugnehmen auf Steuerungsdaten und
- das Formulieren der Varianz (länder-, einstellungs- und situationsspezifische Anpassung des Verhaltens und des Erscheinungsbilds der Oberfläche).

Im Folgenden bezeichne ich diese Fragestellungen als ***Standardprobleme*** oder ***typische Phänomene*** der UI-Entwicklung.

Man kann die typischerweise beim Entwickeln einer größeren Softwareoberfläche auftretenden Standardprobleme wie folgt klassifizieren.

- ***Formulieren der Ablaufbeschreibung***
  - Wie gliedert man die Abläufe? Wie grenzt man die Ablaufschritte voneinander ab?
  - Wie identifiziert und kennzeichnet man kanonische Sequenzen, Ablaufalternativen und parallele Abläufe?
  - Wie identifiziert und kennzeichnet man im Ablauf Haupt- und Nebenpfade?
  - Wie integriert man die Fehler- und die Ausnahmenbehandlung in die Ablaufdefinition?
- ***Wieder verwenden und vererben***
  - Wie kann man Dialogelemente, Dialogseiten und Ablaufmechanismen als wieder verwendbare Bausteine verwenden? Wie formuliert man Gemeinsamkeiten und Unterschiede zwischen dem Muster und seiner spezialisierten Wiederverwendung („same-but-different-problem“)?
  - Wie vererbt man Festlegungen quer durch das UI-Modell (z.B. Hintergrundfarbe, Systemmeldungen)?
- ***Transitionen definieren***
  - Formulieren von situationsabhängigen impliziten (z.B. Weiter-Button) und expliziten Übergängen zwischen Dialogseiten (Workflow-Führung) und Kontrollelementen (Fokus-Führung).
  - Abbilden von Entscheidungstabellen und Verzweigungsknoten auf Bedienungselemente (Zum Beispiel: Mit welchen Elementen setzt man eine mehrstufige Auswahl um?).
- ***Kontextabhängige Präsentationslogik***
  - Anzeige- und Navigationsform für Massendaten (z.B. umfangreiche Listen, Kataloge und Verzeichnisse).
  - Auswerten von komplexen zusammengesetzten Systemsituationen am UI.
- ***Lokalisieren***
  - Anpassen von Texten, Grafiken, Sprachausgaben, Formatierung an lokale Begebenheiten (z.B. Ländereinstellungen, Spracheinstellungen, Skinning).
  - Anpassen der Menüs, Dialogseiten und Abläufe an Konfigurationen (z.B. keine Funktionen anzeigen, die

in dieser Konfiguration oder von dieser Benutzerrolle nicht ausgeführt werden können).

Standard-Entwicklungswerkzeuge (z.B. Java in Eclipse oder eine User Interface-Beschreibung in Form eines strukturierten Anforderungsdokuments) unterstützen das Handling des oben aufgezählten Standardrepertoire der Spezifikationsprobleme schwach. Typischerweise muss sich jeder Entwickler selbst ein Schema oder eine Konfigurationsdatei erfinden, um die UI-Definition zu strukturieren oder zu parametrisieren. Die Lösung wird dabei so gut, wie die ad hoc und neben der eigentlichen Entwicklungsaufgabe gefundenen Strukturen und Beziehungen sind.

Weitere typische - jedoch seltener auftretende - Standardprobleme der UI-Entwicklung sind:

- ***Transaktionsverarbeitung***
  Erst wenn alle Eingabedaten vorliegen, kann die Transaktion ausgeführt werden. Die Eingabedaten müssen (gegebenenfalls über eine lange Sequenz von Dialogseiten hinweg) überprüft werden. Es müssen Berichtigungsanweisungen für die Eingaben ausgegeben werden. Ggf. soll der Anwender im UI zu den zu berichtigenden Stellen automatisch hinnavigiert werden.
- ***Einzelaktionsverarbeitung***
  Einzelne Aktionen in der Benutzeroberfläche sollen sofort verarbeitet werden, und an der Oberfläche soll eine Reaktion sichtbar sein. Das direkt manipulierte oder über eine Eigenschaftenliste (Properties) oder über Kontrollelemente indirekt manipulierte Objekt soll sofort Veränderungen anzeigen.
- ***Benutzerkollaboration***
  Mehrere Anwender bearbeiten z.B. einen gemeinsamen Text oder an einem gemeinsamen Verwaltungsvorgang. Verschiedene Notifikationen sollen sicherstellen, dass die Anwender wissen, was jeweils der Andere tut. Einige Dialoge können nur lesend abgerufen werden, solange sie von einem anderen Anwender bearbeitet werden (Locking).

UI-Perspektiven helfen beim Handling typischer Phänomene des UI-Entwickelns.

UI-Perspektiven liefern den Schlüssel zum systematischen Handling der typischen Spezifikationsprobleme. Prinzipiell ist die Anwendung der UI-Perspektiven als Strukturierungsmethode in jeder technischen Umgebung (z.B. eingebettet in Java oder als UML-Profil) möglich.

Entscheidend für die Qualität eines UI-Werkzeugs ist der Grad der direkten Unterstützung des Modellierens eines UI aus verschiedenen Perspektiven. Daher werden die Informationsstrukturen für eine UI-Spezifikation im späteren Verlauf des Buchs nicht in eine objektorientierte Programmiersprache eingebettet.

Es wird stattdessen eine spezielle Spezifikationssprache vorgestellt und eine Methode vorgeschlagen, die auf das Modellieren eines UI aus verschiedenen Perspektiven ausgelegt ist.

3.2 Anforderungen an UI-Werkzeuge

Warum brauchen wir Anforderungen an Werkzeuge zur UI-Entwicklung?

Klar definierte Anforderungen an Werkzeuge zur UI-Entwicklung sind unabdingbar, um zu bewerten, was Spezifikations-, Implementierungs- und Testwerkzeuge können müssen, damit man mit ihnen User Interfaces entwickeln kann.

Halt, werden Sie sagen: Mit vorhandenen Werkzeugen kann man ja Anwendungsoberflächen entwickeln. Was ist denn daran nicht in Ordnung? Eine berechtigte Frage, auf die ich im Verlauf dieses Kapitels im Detail eingehe.

Die gewünschten Eigenschaften von Werkzeugen zum Erstellen von Benutzeroberflächen können in Anlehnung an Shneiderman [Shneiderm. 2002, S.209] wie folgt in einer groben Übersicht zusammengestellt werden.

- Unabhängigkeit des UI
  - Trennung der UI-Entwicklung von den „inneren Eigenschaften“
  - Möglichkeit verschiedener Strategien für User Interfaces
  - Möglichkeit des Multiplattform Supports
  - Etablieren der Rolle des UI-Entwicklers
  - Durchsetzen von Standards
- Methodik und Notation
  - Entwickeln von Design-Prozeduren
  - Wege, über das Design zu sprechen
  - Schaffen von Projektmanagement für das UI
- Schnelle Entwicklung von Prototypen
  - Frühes Ausprobieren von Ideen
  - Testen, überarbeiten, testen, überarbeiten, ...
  - Beteiligung von Anwendern, Managern und Kunden
- Softwaresupport
  - Steigern der Produktivität

  - Überprüfen der Grenzen und der Konsistenz
  - Team-orientierte Herangehensweise fördern
  - Pflege vereinfachen

Auch wenn diese Aufstellung mehr die Ziele der Verwendung von UI-Werkzeugen als deren tatsächliche Features adressiert, liefert sie eine Ausgangsbasis für das Formulieren konkreter Anforderungen.

Anhand der Use Cases bei Spezifikation, Realisierung und Test von Benutzeroberflächen (siehe auch Abschnitt „Use Cases des UI-Modellierens“ im Kapitel „Sprache“) werden aus Sicht der UI-orientierten Architektur folgende Anforderungen gestellt.

Vokabular zur UI-Spezifikation

- ***Vokabular*** zur Spezifikation von Ablaufstruktur, Bedienungselementen, Layout, Interaktionen sowie Kontext- und Anforderungsbezügen der Benutzerschnittstelle.
- UI-Definition Stufenweise detaillieren
- Möglichkeit, die UI-Spezifikation ***stufenweise*** von einer groben Skizze zu einer Umsetzungsvorschrift zu detaillieren.

UI-Bestandteile kennzeichnen

- Möglichkeit, Bestandteile der Spezifikation bezüglich
  - Fertigstellungsphase,
  - Fertigstellungsgrad (in der jeweiligen Phase),
  - Review-Status (im jeweiligen Fertigstellungsgrad)

  auf beliebiger Gliederungsebene zu ***kennzeichnen***.

Spezifikationsstände vergleichen

- Unterstützen von ***Vergleichen*** zwischen Spezifikationsständen, z.B. Delta-Analyse, Kennzeichnen und Wiederherstellen eines Spezifikationsstandes.

Änderungen verwalten und nachvollziehen

- Unterstützen der ***Änderungsverwaltung***, z.B. Änderungen auf Grund von Change-Requests (Änderungsanforderungen).
  - Nachvollziehen des Ersetzens von Spezifikationsteilen durch andere (replaced-by-Bezug).
  - Nachvollziehen des Wegfalls von Spezifikationsteilen (deprecated-removed-reviewed-Kennzeichnung).
  - Nachvollziehen des Hinzufügens von Spezifikationsteilen (added-to-Bezug).

Gemeinsam an einer UI-Beschreibung arbeiten

- Unterstützen des ***gemeinsamen*** Arbeitens an der Entwicklung einer Benutzeroberfläche.
  - Aufteilbarkeit nach UI-Teilen (z.B. Modul, Screens, Variantenzuschnitt).

  - Aufteilbarkeit nach Informationssegmenten (z.B. Kontext, Screens, Ablaufstruktur, Interaktionen).
  - Aufteilbarkeit nach Fertigstellungsphase und Rolle (z.B. Rahmenspezifikateur, Detailspezifikateur, Realisierer, Tester).

- Werkzeugunabhängige ***Zugreifbarkeit*** auf die spezifizierten Informationen als menschen- und als maschinenlesbare Datenstruktur (in Textform).

  Unabhängig vom Werkzeug auf Spezifikationsdaten zugreifen

  - Definiertes Informationsmodell
  - Bereitstellung als Klartextdatei, die zum Informationsmodell konform ist (z.B. als HMIML, Konkreto-Skript oder im projekt-spezifischen XML-Format).
  - Kapselung von werkzeugabhängigen [fachlichen] Informationen der Spezifikation und von technischen Implementierungsrestriktionen.

- Unterstützung von verschiedenen ***Sichten*** auf die spezifizierten Inhalte (auf Basis der werkzeugunabhängigen Zugreifbarkeit)

  Spezifikationsdaten auswerten und umformen

  - Auswertungen
  - Umformungen
  - Simulation
  - Generieren von Steuerungsdaten
  - Generieren von Programmcode

  mit wahlfreien Analyse- und Bearbeitungswerkzeugen.

3.3 Bewertungsschema

Im Mindestumfang besteht eine UI-Entwicklungsumgebung aus einem ***User Interface Editor***, der die Modellierung von UI-Eigenschaften mit einer bestimmten ***Modellierungsnotation*** ermöglicht. Die Notation kann z.B. aus grafischen Symbolen für Kontrollelemente, aus Property-Tabellen, Code-Skripten oder aus grafischen Modellen bestehen.

Shneiderman formuliert die Ziele einer Notation so [Schneiderm 2002, S.325f]:

- Grundlegende Ziele
  - Präzision
  - Kompaktheit
  - Leichtigkeit beim Schreiben und Lesen
  - Vollständigkeit

  - Geschwindigkeit beim Lernen
  - Einfachheit, Fehler zu vermeiden
  - Leichtigkeit, sich an das Notierte zu erinnern
- Ziele auf höherer Abstraktionsstufe
  - Enge Korrespondenz zwischen Realität und Notation
  - Bequemlichkeit beim Durchführen von Manipulationen, die für Notationsanwender von Bedeutung sind
  - Kompatibilität mit anderen Notationen
  - Anpassbarkeit an Anfänger und Experten
  - Ausdruckskraft, die die Kreativität fördert
  - Visuelle Ansprechbarkeit („Anziehungskraft“)
- Einschränkende Ziele
  - Fähigkeit des Menschen, in der Notation zu schreiben und zu lesen
  - Zusammenpassen der Aufzeichnungs- und der Anzeigemedien
  - Bequemlichkeit beim sich Ausdrücken in der Notation („Aussprache“)

UIEU

Diese Ziele gelten auch für das Verwenden der Notation in einer User Interface-Entwicklungsumgebung (UIEU).

Die Form der Notation ist im nachfolgenden Bewertungsschema für UI-Entwicklungsumgebungen sekundär. Das Bewertungssystem klassifiziert die Tätigkeiten eines Entwicklers bzw. Spezifizierers beim Verwenden einer UI-Entwicklungsumgebung und stellt Bewertungskriterien für die Qualität, mit der die UIEU diese Tätigkeiten unterstützt.

Das macht der Anwender mit einer UI-Entwicklungsumgebung:

- ***Inkrementieren***: Hinzufügen von User Interface-Bestandteilen (z.B. Steps, Dialog, Lists, List Items).
- ***Transkribieren***: Ausdetaillieren und Ausformulieren von User Interface Screen-Content, Verhalten, und Kontext; verknüpfen mit Anforderungen.
- ***Modifizieren***: Ändern von User Interface-Bestandteilen zur Anpassung an veränderte Anforderungen.
- ***Exploratives Design***: Skizzieren; iteratives, entdeckendes Vervollständigen von User Interface-Bestandteilen.
- ***Exploratives Lesen***: Erforschen der Struktur, der Klassifikation, der spezifizierten User Interface-Bestandteile.

- ***Suchen***: Nachforschen nach bekannten Fragestellungen, z.B. welche User Interface-Bestandteile haben Interaktionen.
- ***Kommunizieren***: Präsentation und Kommunikation der UI-Entwürfe im Entwicklungsteam und vor dem Kunden.

Die Tätigkeiten und die Kriterien sind an die Arbeit von Blackwell und Green über das Bewerten von Notationen in [Blackwell 2003] angelehnt. Das Schema wurde von mir an die Bewertung von UI-Entwicklungswerkzeugen angepasst, erweitert und mit einem Punktesystem versehen.

Bewertungskriterien für eine UI-Entwicklungsumgebung sind:

- ***Vollständigkeit***:
  Abdeckungsgrad der UI-Domäne
- ***Fachlichkeit***:
  Nähe der Repräsentation zur UI-Domäne
- ***Sichtbarkeit***:
  Kapselungsgrad von Informationen
- ***Geringe Vorabverpflichtung***:
  Grad der Provisionalität (Vorläufigkeit) der Definitionen
- ***Klarheit***:
  Umfang, Erkennbarkeit und Lesbarkeit der Konstrukte
- ***Geringe Zähigkeit***:
  Widerstand gegen Änderung der UI-Beschreibung

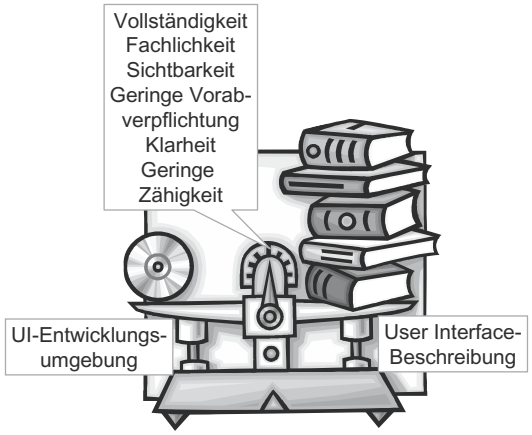


Bild 27: Bewertungskriterien für eine UI-Entwicklungsumgebung

Für jedes der nachstehend beschriebenen Bewertungskriterien können 1 bis 10 Punkte vergeben werden. Die Gesamtzahl der Punkte ist eine Maßzahl für die Güte einer UIEU.

Vollständigkeit

Abdeckungsgrad des UI-Datenmodells

Ein Informations- bzw. Datenmodell für das Entwickeln von User Interfaces stellt den Rahmen für das Wissensmanagement von Informationen und Beziehungen, die zum Bau eines UI dienen. Die Notation zur UI-Modellierung ermöglicht es, dieses Datenmodell mit Inhalten zu befüllen.

Je höher der Abdeckungsgrad des UI-Datenmodells, desto vollständiger lässt sich mit der Notation ein UI (einschließlich der Verwaltung der Beschreibung selbst, z.B. Varianten, Spezifikationsstände, Änderungen) beschreiben.

Die Vollständigkeitsbewertung erfolgt auf einer Skala von 1 bis 10 Punkte. Die Punkte werden anhand des Abdeckungsgrads der UI-Perspektiven und ihrer Unterobjekte durch die bewertete UI-Entwicklungsumgebung vergeben.

- 2 Punkte: Werkzeug deckt die Dialogperspektive ab
- 4: Werkzeug deckt zusätzlich die Kontextperspektive ab
- 6: Werkzeug deckt zusätzlich die Strukturperspektive ab
- 8: Werkzeug deckt zusätzlich die Interaktionsperspektive ab
- 10: Vollständige Abdeckung der UI-Perspektiven (einschließlich der Anforderungsperspektive), der UI-Objekte und der Beziehungen zwischen ihnen.
- Für die fragmentarische Abdeckung einer Perspektive kann z.B. jeweils ein Punkt weniger vergeben werden.

Fachlichkeit

Nähe zur Fachsprache der Anwender

Ist die Notation nahe an der Sprache, die zu Fachgesprächen zwischen Anwendern, Spezifizierern und Entwicklern verwendet wird, dann lässt sich das UI in ihr leicht formulieren und das Formulierte leicht lesen.

Fachlichkeit bedeutet, dass UI-Elemente und ihr Verhalten durch ein Vokabular beschrieben werden, das nahe an dem aus Sicht der UI-Bedienung durch den Anwender verwendeten Vokabular liegt.

Die Fachlichkeitsbewertung erfolgt anhand der Bezeichnungen für die Hauptobjekte der Modellierungsnotation.

- 1 Punkt: Konstruktionsorientiertes Vokabular
- 10 Punkte: Anwenderorientiertes Vokabular

Sichtbarkeit

Möglichst einfache Struktur

Eine UI-Entwicklungsumgebung, die Informationen kapselt und vielfach verpackt, reduziert die Sichtbarkeit von UI-Eigenschaften.

Bei Pflege von UI-Details, z.B. bei Anzeigetexten, ist eine hohe Sichtbarkeit erforderlich.

Eine geringe Sichtbarkeit ist vor allem dann gegeben, wenn die Eigenschaften z.B. in einem geschachtelten Propertybaum stecken, in dem man durch Aufklappen nach der Stelle suchen muss, in der eine bestimmte Eigenschaft definiert ist.

Man kann dieses Manko teilweise durch eine Suchfunktion des Editors beheben, hat aber nach wie vor Sichtbarkeitsprobleme, wenn man ähnliche Properties an mehreren Stellen des Baumes miteinander vergleichen möchte.

Die Sichtbarkeitsbewertung erfolgt anhand der im Editor verfügbaren Gesamt- und Detailsichten auf das Modell sowie anhand der Schachtelungstiefe von Informationen. Die Leitfrage zur Bewertung kann z.B. sein: Wie viele Wrapper werden um die tatsächliche UI-Eigenschaft gelegt?

- 1 Punkt: Informationen werden tief klassifiziert geschachtelt
- 10 Punkte: Informationen werden mit wenig Schachtelungen spezifiziert

Vorabverpflichtung

Toleranz von Spezifikations-Auslassungen

Vorabverpflichtung passiert, wenn man frühzeitig gezwungen wird, Dinge festzulegen, die eigentlich erst später entschieden werden sollen oder können.

Beispiele: Id's für Elemente festlegen müssen, die vielleicht gar nicht referenziert werden. Für eine Auswahlliste sofort festlegen, ob sie über eine Drop-Down-Liste oder über eine Menütafel realisiert wird.

Provisionalität erlaubt unvollständiges Spezifizieren, Standard-Annahmen zu unspezifizierten Inhalten und spätere Konkretisierung, wenn nötig.

Die Bewertung der Provisionalität erfolgt anhand der Striktheit der Notation. Das kann anhand folgender Leitfragen bewertet werden: Können Teile von Notationskonstrukten zunächst weggelassen werden? Kann das UI von einer groben Skizze an stufenweise zu den Details hin spezifiziert werden?

- 1 Punkt: Der UI-Editor lässt keine Auslassungen von Eigenschaften und keine vorläufigen Definitionen zu.

- 10 Punkte: Notation erlaubt unvollständige und vorläufige Konstrukte, die nach und nach exakter ausformuliert werden können.

Klarheit

Möglichst knappe, leicht voneinander zu unterscheidende Formen

Neigt eine Notation zu verschiedenen Beschreibungskonstrukten für die gleiche Sache, nimmt sie auf dem Bildschirm unverhältnismäßig viel Platz für einfache Informationen in Anspruch oder benutzt sie lange, umständliche und leicht zu verwechselnde Konstrukte, dann hat sie eine hohe Diffusität. Es heißt nichts anderes, als dass die eigentliche Information in der Beschreibung verstreut ist.

Hohe Diffusität führt zu schlechter Formulier- und Lesbarkeit der Notation und erhöht damit den Kommunikationsaufwand der am UI-Entwurf und Design beteiligten Rollen.

Die Bewertung der Diffusität erfolgt anhand der Länge und der Verwechselbarkeit der Notationskonstrukte. Sind viele Notationsworte für eine Festlegung notwendig? Können ähnliche Konstrukte leicht miteinander verwechselt werden?

- 1 Punkt: Ähnliche Notationselemente haben unterschiedliche Bedeutungen
- 10: Elemente sind klar voneinander unterscheidbar

Zähigkeit

Die Spezifikation soll schnell und einfach zu ändern sein.

Eine viskose UI-Entwicklungsumgebung bedarf vieler Anwenderaktionen, um ein Ziel zu erreichen.

Zum Beispiel kann das Ändern aller Step-Namen einer Ablaufstruktur von Groß- in Kleinschreibung das Anfassen eines jeden Steps bedeuten.

Editierumgebungen, die passende Abstraktionen (zum Beispiel Suchen und Ersetzen in allen Step-Namen) bereitstellen, können Viskosität senken.

Die Viskositätsbewertung erfolgt anhand des Aufwands zum Ändern von Modellkonstrukten. Mögliche Leitfragen für diese Bewertung sind: Muss man das Modell an mehreren Stellen ändern, um ein Element hinzuzufügen oder zu entfernen? Kann ein Teil eines Modells durch ein anderes Teil ersetzt werden? Gehen beim Verschieben von Modellteilen Beziehungsinformationen verloren?

- 1 Punkt: Bei Änderungen der UI-Beschreibung gehen Entitäten und Relationen verloren

- 10 Punkte: Die UI-Beschreibung kann beliebig ohne Informationsverlust verändert und Notationselemente können schnell umgeformt werden (z.B. aus einem Pushbutton eine Checkbox machen)

Gesamtbewertung

Die Gesamtbewertung kann durch die Summe der in den einzelnen Bewertungskriterien vergebenen Punkte ausgedrückt werden.

Die maximale Punktzahl beträgt 60 Punkte. Ein mit 60 Punkten bewertetes Werkzeug wäre nach diesem Bewertungsschema optimal zur UI-Entwicklung geeignet.

Das folgende Beispiel zeigt eine Gesamtbewertung im Vergleich zwischen C++ mit MFC und Flash MX. Die Einzelbewertungen sind subjektive Schätzwerte.

	Punkte	
Bewertungskriterien	C++ mit MFC	Flash MX
Vollständigkeit	10	6
Fachlichkeit	2	4
Sichtbarkeit	2	6
Freiheit von Vorabverpflichtung	4	6
Klarheit	4	6
Geringe Zähigkeit	4	4
Gesamtbewertung (Summe)	26	32

Bild 28: Beispiel für eine Bewertung von UI-Werkzeugen.

Die im Beispiel mit 26 und 32 Punkten bewerteten Werkzeuge liegen bezüglich ihrer Eignungseinschätzung im Mittelfeld.

Auf eine detaillierte Bewertung von einzelnen Werkzeugen wird im Rahmen des Buchs bewusst verzichtet, da solche Bewertungen stark vom Kontext des durchzuführenden Projektes und der Aufgabenstellung der zu erstellenden Software abhängen. Eine allgemeingültige Bewertung eines Werkzeugs ohne das Projektumfeld im Detail zu berücksichtigen, erscheint mir wenig plausibel und kann sogar irreführend sein. Ich empfehle Ihnen daher, die für Ihr Projekt in Frage kommenden Werkzeuge fallbezogen und subjektiv anhand des obigen Schemas einzuschätzen.

3.4 Werkzeugarten

Zum Beschreiben des User Interface wenden die Entwickler in der Projektpraxis die unterschiedlichsten Hilfsmittel an.

Die von mir beobachteten UI-Entwurfsmethoden reichen von Flipcharts, mit denen Workshopräume tapeziert wurden, über Paintshop, Powerpoint, Excel und HTML-Editoren bis hin zu GUI Toolkits wie FormFlow, QT oder VisualBasic Designer oder State Charts und Spezialeditoren für die UI-Modellierung.

Zu unterscheiden sind:

- Office-Werkzeuge (z.B. Powerpoint und Excel)
- Darstellungsorientierte Werkzeuge (z.B. Flash, Director, SVG oder (X)HTML)
- GUI-Toolkits (Betriebsystemnahe und Kontrollelement-orientierte Designwerkzeuge und Bibliotheken)
- Grafische Ablaufmodellierungswerkzeuge (oft auf Basis von State Charts)
- Werkzeuge für eingebettete Systeme (z.B. Guide oder HMI Studio)
- Domänenspezifische Modellierungssprachen

3.5 Office-Werkzeuge

Office-Programme z.B. Word, Excel, Powerpoint werden vor allem von Anwendervertretern als Werkzeuge für den Entwurf von User Interfaces bevorzugt. Das liegt in den meisten Fällen daran, dass sie sofort verfügbar sind und dass man sich nicht erst einarbeiten muss, sondern sofort beginnen kann, eine Maske zu entwerfen oder einen Ablauf zu skizzieren. Man muss mit Office-Programmen keine bestimmte Methodik befolgen.

In der Praxis hat es sich eingebürgert, dass Powerpoint zum Entwurf von Masken und Abläufen verwendet wird. Excel wird oft dazu benutzt, Listen von Kontrollelementen oder Dialogschritten zu erstellen. In der Regel werden von Analytikern projektspezifische Vorlagen (Templates) für den Entwurf von Maskeninhalten und für die Beschreibung von Kontrollelementen aufbereitet und den Anwendervertretern vorgegeben.

Je nach Detaillierungsgrad kann eine Spezifikationsvorlage die gewünschte Struktur der UI-Beschreibung grob umreißen oder bis in die Einzelheiten (z.B. Wertebereiche von einzelnen Eingabefeldern) vorschreiben.

Eine Überprüfung der formalen Korrektheit der auf Basis einer Word-Vorlage erstellten UI-Beschreibung findet nicht statt. Eben-

so wenig wird der Spezifizierer programmseitig beim Schreiben der UI-Spezifikation unterstützt (z.B. auf formale Fehler hingewiesen oder zum Einhalten der vorgegebenen Struktur aufgefordert).

Zur besseren Durchsetzung von vorgegebenen Spezifikationsschablonen werden Anforderungsmanagementsysteme wie Doors oder Rochade eingesetzt. Der Spezifikationsart nach können diese Programme zu den Office-Werkzeugen gezählt werden, weil die Spezifikation aus Textbausteinen erstellt wird. Zusätzlich wird über die Textbausteine eine Struktur gelegt, deren Verwaltung und Einhaltung mit Hilfe des Anforderungsmanagement-Programms erfolgt.

3.6 Darstellungsorientierte Werkzeuge

Dazu zählen Werkzeuge, bei denen die grafische Gestaltung und Animation, also die Form des User Interface im Vordergrund steht. Beispiele: Flash, SVG (Scalable Vector Graphics), XHTML.

Darstellungsorientierte Werkzeuge werden von Grafikdesignern bevorzugt und meist dazu eingesetzt, die grafische Gestaltung von Dialogseiten festzulegen.

Beim Arbeiten mit darstellungsorientierten Werkzeugen wird zunächst die grafische Erscheinung von Bedienelementen und das Layout der Dialogseite festgelegt, bevor eventuelle Interaktionen hinzugefügt werden.

3.7 GUI-Toolkits und GUI-Bibliotheken

Es gibt eine Reihe von GUI-Toolkits, mit denen ***Screendesign*** möglich ist. Viele Entwicklungswerkzeuge, etwa .Net Visual Studio oder Delphi beinhalten eigene Screen Editoren, mit denen Bildschirmmasken aus dem Vorrat von Kontrollelementen „zusammengeklickt“ werden können.

GUI-Bibliotheken (APIs) sind Funktionssammlungen, die dem Programmierer das Codieren von grafischen Benutzeroberflächen erlauben.

PC-typische GUI-Toolkits bzw. GUI-Bibliotheken sind: XWindows, Windows SDK, Motif SDK, GTK und MacOS. Eine detaillierte Beschreibung der typischen Bestandteile eines GUI-Toolkits finden Sie z.B. in [Lauer 2002, S.56-76].

Es gibt kaum GUI-Toolkits, die über das Screendesign hinausgehen und beispielsweise die Modellierung des ***Verhaltens*** der Oberfläche ermöglichen. Ebenso selten sind GUI-Toolkits, mit denen eigene Kontrollelemente und Abhängigkeiten des Er-

scheinungsbildes der Kontrollelemente vom Ablaufkontext der Anwendung modelliert werden können.

Sowohl Verhalten als auch Kontextbezug werden beim Einsatz von GUI-Toolkits bzw. GUI-Bibliotheken typischerweise in der Programmiersprache kodiert, für die die Bibliothek erstellt wurde.

3.8 Grafische Ablaufmodellierungswerkzeuge

Ein Konsens über den Modellierungsbedarf im Bereich der Anwendungsoberflächen kristallisierte sich nur langsam heraus. Die von David Harel Mitte der 80ger Jahre vorgeschlagene State Charts Notation [Harel 1987] trug den Verschiebungen von prozeduralen zu reaktiven Systemen Rechnung und lieferte erstmals eine standardisierte Beschreibungsform für Teile von Oberflächen.

Die State Charts können die ***Ablauf***- und die ***Strukturaspekte*** einer Oberfläche beschreiben, bieten aber keine Hilfe zum Beschreiben, wie Dialoge und Anwender-Notifikationen dargestellt werden. State Charts bieten auch keine explizite Schnittstellensicht zum funktionalen Teil der Anwendung.

Dennoch spielen Statecharts nach wie vor eine zentrale Rolle für die Ablaufmodellierung von User Interfaces. State Charts werden von verschiedenen Experten als Instrument zur UI-Entwicklung diskutiert und empfohlen.

Im Kapitel Sprache, im Abschnitt „State Charts Notation“ finden Sie eine Einführung in die State Charts Notation sowie eine Analyse der Verwendbarkeit von State Charts zur Spezifikation von Oberflächen.

State Charts-Editoren werden oft in Werkzeugen zur formalen UI-Modellierung integriert. DSL-Editoren zur Modellierung von UI-Abläufen haben meist das Metamodell der State Charts als Basis. Auch in integrierten Werkzeugen für eingebettete Systeme sind State Charts ein typischer Bestandteil der UIEU.

3.9 Werkzeuge für eingebettete Systeme

Im Folgenden werden zwei Werkzeuge vorgestellt, die die gesamte Oberfläche im Blickwinkel der Modellierung behalten. Sie unterscheiden sich in Komplexität, Philosophie und Bedienung grundlegend von den üblichen Maskeneditoren.

Beide Werkzeuge bringen einen jeweils eigenen Modellierungsansatz mit, mit dem ein Gesamtmodell eines UI erstellt, zur Mo-

dellierungszeit simuliert und Code für die Zielplattform erzeugt werden kann.

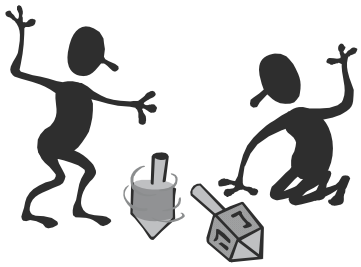


Bild 29: Durchgängige UI-Werkzeuge haben eigene Philosophien

Das originäre Einsatzgebiet dieser Werkzeuge liegt in der Entwicklung von Software für eingebettete Systeme in Fahrzeugen.

Moderne Infotainmentsysteme der Automobilindustrie zeichnen sich durch die Integration verschiedener Geräte (Radio, CD, CD-Wechsler, MP3, DVD, TV, Navigation, Verkehrsinformationen, Telefon etc.) in einem User Interface aus. Dabei spielen ein durchgängiges Bedienkonzept und Erscheinungsbild Benutzeroberfläche eine entscheidende Rolle.

Die Kurzanalyse der Werkzeuge zeigt die Bandbreite der Modellierungsthemen und die Möglichkeiten, wie diese Themen (siehe auch Abschnitt „Typische Phänomene des UI-Entwickelns“ in diesem Kapitel) durch Features unterstützt werden können. Zu dieser Analyse gehören:

- Beschreibung des Einsatzzwecks und der Zielgruppe
  - Was wird mit den Werkzeugen entwickelt?
  - Welches Vorgehensmodell sieht das Werkzeug vor?
  - Welche Rollen verwenden das Werkzeug?
  - Welche Ergebnisse liefert das Werkzeug?
  - Welche Kenntnisse und Vorbedingungen werden vorausgesetzt?
- Beschreibung der Features des Werkzeugs
  - Welche Instrumente werden vom Werkzeug für welchen Zweck bereitgestellt?
  - Welche Anwendungsfälle werden unterstützt? Für welche Anwendungsfälle ist das Werkzeug nicht vorgesehen.
- Ein Beispiel für das Modellieren mit dem Werkzeug (in Form von kommentierten Screenshots)

Die beim Modellieren von UIs für eingebettete Systeme eingesetzten Methoden können ohne weiteres auf herkömmliche Büroanwendungen übertragen werden.

3Soft tresos Guide[1]

Guide ist ein Werkzeug zur Entwicklung eingebetteter Softwareoberflächen. Das User Interface wird grafisch in einer IDE (Intergrated Development Environment) mittels spezialisierter Editoren (z.B. View-Editor für Bildschirme, State-Editor für die Ablaufstruktur) modelliert.

Das Werkzeug ermöglicht die Simulation des UI als Prototyp und das Generieren von Teilen des Codes für die jeweilige Zielplattform aus den Modelldaten.

3soft tresos Guide

Einsatzziele und Zielgruppe

- Modellbasierte Spezifikation und Simulation grafischer Oberflächen.
- On-the-fly-Simulation während der Spezifikationsphase.
- Erstellen eines „lebenden Lastenhefts“ in der frühen Phase des Entwicklungsprozesses.
- Integrierte Entwicklung von GUI und Sprachdialog in einem Werkzeug; Verzahnung von Modalitäten (multimodaler Dialog).

Übersicht der Features

- WYSIWYG (What You See Is What You Get) View-Editor zur Erstellung der einzelnen Bildschirme – „Views“
- Editor für UML-Zustandsdiagramme zur Beschreibung des Menüablaufs
- Anbindung von Spracherkennern und Text-To-Speech-Systemen
- Integrierter Simulator zum prototypischen Ausprobieren der spezifizierten Oberfläche.
- Aufzeichnen und Playback von Bedienabläufen in der Simulation
- Einbindung echter Hardware in der Simulation
- Projektspezifische Erweiterung durch Plugins
- Anbindung von projektabhängigen Java Widgets
- Multi-User-fähig für paralleles Arbeiten an verschiedenen Teilen der Spezifikation
- Automatische Konsistenzüberprüfungen des Modells

[1] Darstellung auf Basis der Angaben des Herstellers (mit freundlicher Genehmigung der 3Soft GmbH, Erlangen) und eigener Tests.

- Speicherung in maschinenlesbarem XML
- Spezifikation mehrerer Zustandsmaschinen für mehrere Displays oder Splitscreens
- Der Variantenmechanismus erlaubt die Definition verschiedener Skins z.B. für Tag/Nacht-Design
- Unterstützung zur Internationalisierung
  - Eigenschaften wie Texte, Bilder, Fonts, Koordinaten können abhängig von der Sprache definiert werden
  - Automatische Textlängenberechnung
  - Export / Import-Funktion für das Übersetzungsbüro
- Referenzsuche „Wo wird dieses Element in der Spezifikation verwendet“
- Erstellung von Templates, um bereits konfigurierte Widgets wieder zu verwenden
- Papierkorb-Konzept
- Export der Spezifikation
  - in spezielle XML-Formate
  - als PDF-Dokumentation
- Batch-Screenshot aller Views
  - Als Bitmap
  - Als Vektorgrafik (SVG)
- Codegenerierung für eingebettete Systeme
  - für verschiedene Betriebssysteme, embedded Grafikbibliotheken und Programmiersprachen z.B. VxWorks, WinCE, QNX, Linux, C, C++, Java / OSGI.

Modellieren mit Guide

Ein „View“ wird im View-Editor spezifiziert. Links oben ist die Baumstruktur der Widgets des aktuell gezeigten Views zu sehen. Rechts in der Mitte wurde gerade die aktuelle Variante „Skin Digital, Day“ ausgewählt.

Die im View-Editor spezifizierten Views werden im Zustandseditor zur Definition des Menüablaufs verwendet. Auf verschiedenen Diagrammen wird das Verhalten des Systems mit UML-Zustandsdiagrammen beschrieben.

Illustrierte Zustandsdiagramme und automatische Überprüfungen erleichtern dabei die Arbeit.

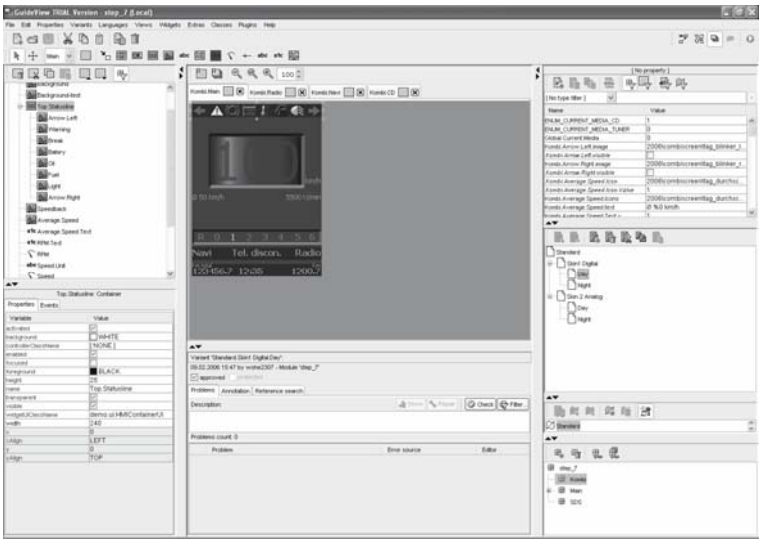


Bild 30: Guide View-Editor

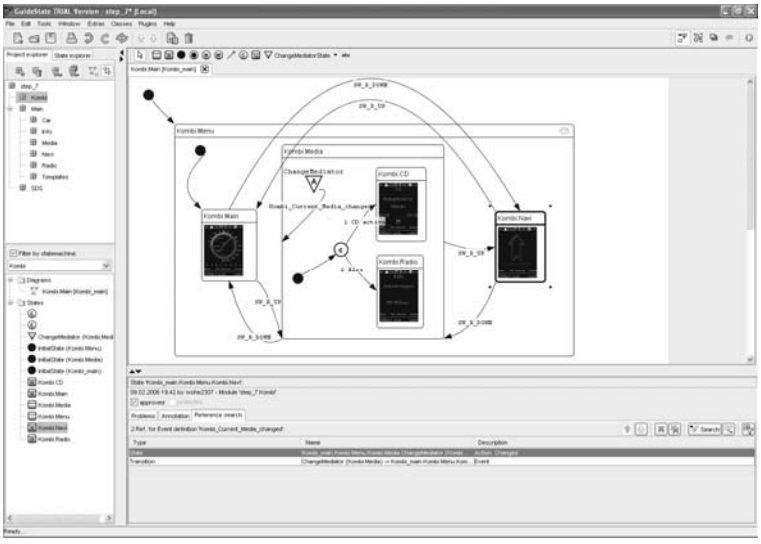


Bild 31: Guide State-Editor

Die Trennung zwischen Oberfläche und Applikation erfolgt durch den Datenpool und über Events. Dies erlaubt eine User Interface-Spezifikation unabhängig von den funktionalen Applikationen.

- Der Datenpool enthält alle für das HMI relevanten Werte.
- Er kann zu Simulationszwecken mit Testwerten beschrieben werden.

IAV Teledrive HMI Studio[2]

Teledrive HMI (Human Machine Interface) Studio ist ein Werkzeug zur Entwicklung von Oberflächen für Infotainmentsysteme. Das UI wird in einer IDE anhand von einem Strukturbaum mittels verschiedener Property-Editoren entwickelt.

Das Werkzeug unterstützt die Verhaltensmodellierung von Widgets, das Simulieren des UI mit einem externen Simulator sowie die Verwendung einer XML-basierten Modellierungssprache als Modellträger.

Einsatzziele und Zielgruppe

- Werkzeugkette für den gesamten Entwicklungsprozess
- Durchgängige Entwicklung in einem Vorgehensmodell und mit einer Werkzeugkette
- Einheitliches Datenaustauschformat für alle Beteiligten, dadurch keine Medienbrüche mehr
- Frühzeitige Verfügbarkeit von Simulationen für entwicklungsbegleitende Tests
- Reduktion von Kommunikationsproblemen zwischen Automobilhersteller und Zulieferer

Übersicht der Features

IAV Teledrive HMI Studio

Die Infotainment Markup Language (IML) definiert eine Beschreibungssprache für die einheitliche Erstellung von XML-Dateien für das HMI eines Infotainmentsystems. Sie enthält die Strukturbeschreibungen für den Aufbau und die Erstellung eines konkreten HMI und bildet außerdem die Grundlage für die Gültigkeitsprüfung der XML Dateien.

Die Features von IML sind:

- ein einheitliches Austauschformat für die in einer Spezifikation enthaltenen Informationen
- die Spezifikation ist mit XML-Werkzeugen in andere Zielformate transformierbar.
- die Spezifikation ist maschinenlesbar
- die Spezifikation ist mit einer gültigen Formatbeschreibung unterlegt

.

[2] Darstellung auf Basis der Angaben des Herstellers (mit freundlicher Genehmigung der IAV GmbH, Berlin) und eigener Tests.

Layout- und Verhaltensdefinition von ***Widgets*** werden in einem Definitionsteil festgelegt und in einem Implementationsteil mit konkreten Werten versehen. Der Definitionsteil enthält Definitionen der benötigten Elemente, Zustände und Eigenschaften eines Widgets. Diese definierten Elemente, Zustände und Eigenschaften werden in dem Implementationsteil angewendet, um das Widget-Verhalten und Aussehen zu konkretisieren. Dazu bietet die Struktur eines Widgets unterschiedliche Möglichkeiten:

- Es können Widget-eigene Eigenschaften mit Daten gefüllt werden
- Alle grafischen Elemente werden im grafischen Bereich des übergeordneten Widget platziert
- Alle grafischen Elemente erhalten zustandsbezogen ihr Aussehen
- Man kann beliebig viele Widgets als Elemente ineinander schachteln
- Für jedes Widget können Funktionen definiert werden
- Jedes Widget enthält einen eigenen Datencontainer mit lokalen statischen und dynamischen Variablen
- Zustandsübergänge werden nach Eingang eines Events in Form einer Nachricht in einer Zustandstabelle behandelt
- Jedes Widget besitzt die Fähigkeit, Nachrichten zu versenden

Die Funktionalität kann durch PlugINs erweitert werden. Da das Tool in .NET umgesetzt wurde, werden dabei .NET-Sprachen wie z.B. Visual Basic, Java, C# unterstützt. Es existieren Schnittstellen zu gängigen Dokumentationsformaten, Testmanagement- und Konfigurationsmanagementsystemen. Ein Codegenerator kann integriert werden.

Modellieren mit Teledrive

Menu Structure Editor

Der Aufbau der Menüstruktur erfolgt mittels Kontextmenü oder per Drag & Drop von Vorlagenelementen in die vorhandene Struktur.

Die Konsistenz der HMI-Struktur wird durch die automatische Anwendung festgelegter Strukturregeln gewährleistet. Das bedeutet z.B., dass in Menünetzen ausschließlich Menüs und weitere Menünetze eingefügt werden dürfen und dass Menüs ihrerseits nur Widgets enthalten können.

Die Attribute eines Menüstruktur-Elementes, wie z. B. Name, Kommentar oder Transitionen, können bearbeitet werden. Dies

geschieht in Eigenschafts-Editoren (Property Editor), welche die Eigenschaften in einer tabellarischen Form zur Bearbeitung präsentieren.

Menu Transition Logic Editor

Generell können bedingte sowie unbedingte Sprünge zwischen Menüstruktur-Elementen definiert werden. Das Hinzufügen und Bearbeiten der Sprünge geschieht in tabellarischer Form.

Für eine solche Transition können die Sprungtypen „Menu“, „Net“ (Netzsprung) oder „Back“ definiert werden. Für die Zielauswahl einer Transition werden ausschließlich alle gültigen Ziele des definierten Sprungtyps angeboten.

Basic Widget Pool

Dieser Pool stellt eine Sammlung von wiederverwendbaren Vorlagenelementen für die Definition der HMI-Struktur dar. Die Poolelemente werden dem Benutzer nach Kategorien gruppiert in einem Palettenfenster angeboten.

Property Editor

Die Bearbeitung von Eigenschaften eines Widgets wird in diesem Editor vorgenommen. Für die unterschiedlichen Eigenschaftstypen wie z. B. Zahlen, Farben, Zeichenketten werden angepasste Eingabemöglichkeiten zur Verfügung gestellt.

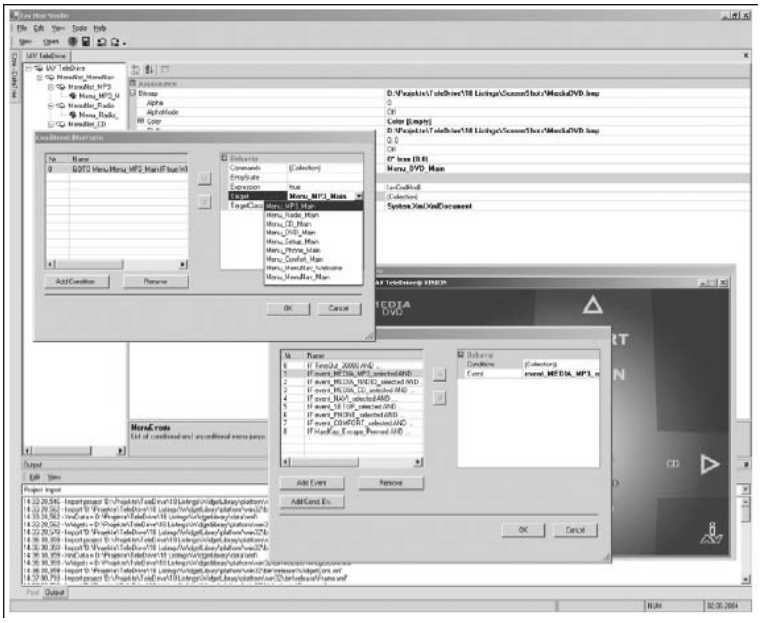


Bild 32: IAV Teledrive - Spezifikation

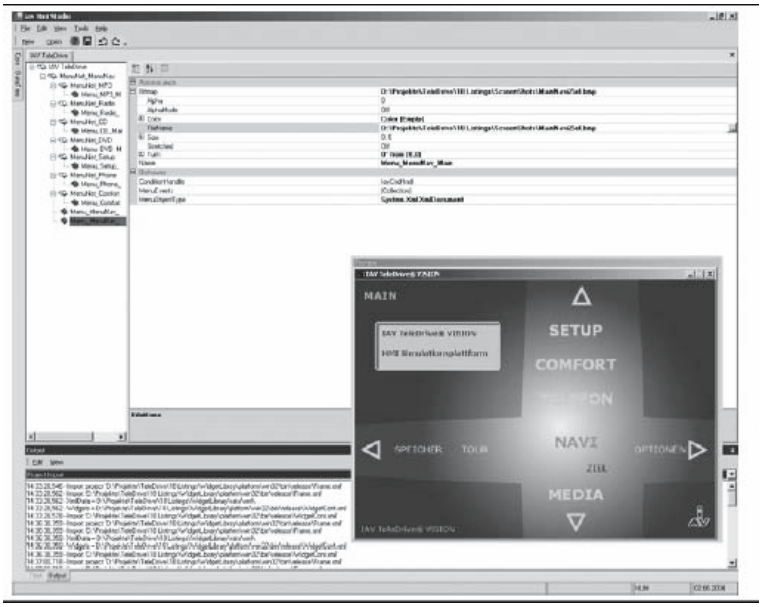


Bild 33: IAV Teledrive - Simulation

Hierarchical Widget Editor

Ein Widget kann aus BaseWidgets und/oder wiederum aus anderen Widgets zusammengestellt werden. Hierfür werden die Elemente des Basic Widget-Pools als Vorlage benutzt. Neu definierte Widgets werden automatisch in einen „User Defined" Widget-Pool für eine spätere Wiederverwendung eingefügt.

Widget Preview

Nach der Zuordnung von Graphikdateien zu Menüs oder Widgets kann der Benutzer über eine statische Vorschau das Erscheinungsbild von Menüs visuell prüfen. Die Zuordnung kann während der Bearbeitungszeit gewählt werden. Die Vorschaudaten werden zusammen mit den Projektdaten persistent gehalten.

Behavior Editor

Dieser Editor dient zur Erstellung und Bearbeitung des Verhaltens von Widgets. Er unterstützt den Entwurf des Verhaltens unter anderem durch Zustandsmaschinen. Darüber hinaus ist es möglich, externe Funktionalität zu integrieren.

Generieren von Dokumenten

Aus dem HMIStudio können projektspezifische Dokumentationen erzeugt werden. Unterstützt werden die Formate DOC (Winword) und HTML.

Resourcen Management

Attributwerte wie z. B. Texte, Positionen, Farben und Bitmaps können durch den Benutzer zu Ressourcen erklärt werden.

Weitere Werkzeuge für das gesamte UI

Neben den vorgestellten Werkbänken zur durchgängigen Entwicklung von Oberflächen für eingebettete Systeme gibt es eine Reihe von experimentellen Projekten und Werkzeugen, welche ebenfalls Lösungsansätze für eine durchgängige UI-Entwicklung anbieten. Darunter gibt es auch kommerzielle Lösungen.

Als Beispiele seien die DSL (Domain Specific Language) Tools von Microsoft (http://msdn.microsoft.com/vstudio/DSL) oder MetaEdit+ von MetaCase (http://www.metacase.com/) genannt.

3.10 Nutzen

Die Analyse von verschiedenen Werkzeugarten und Beispielen für Werkzeuge zur durchgängigen UI-Entwicklung hat uns einen Einblick in die Komplexität der UI-Modellierung sowie einen Überblick über verschiedene Ansätze zur Beherrschung dieser Komplexität gegeben.

Das in diesem Kapitel eingeführte Bewertungsschema für UI-Werkzeuge kann Ihnen bei der Beurteilung und bei der Auswahl des passenden Werkzeugs für Ihr Projekt behilflich sein.

Mit den vorgestellten Anforderungsprofilen an Funktionalität von UI-Entwicklungswerkzeugen und mit den Bewertungskriterien für Handhabung der Werkzeuge können Projektleiter, Architekten und Entwickler die am Markt angebotenen Werkzeuge vergleichen und den Erfüllungsgrad der projektspezifischen Anforderungen einschätzen.

Die Bereitstellung von UI-Entwicklungswerkzeugen ist ein wachsender Markt, und es ist zu erwarten, dass vorhandene Modellierungsansätze weiterentwickelt und neue Werkzeuge angeboten werden. Mit den in diesem Kapitel eingeführten Bewertungsschemen können Sie die Vollständigkeit und die Anwendbarkeit von vorhandenen und neu aufkommenden User Interface-Entwicklungswerkzeugen bewerten.

Über den Vergleich von Werkzeugen hinaus ist das Bewertungsschema für die Entwicklung von User Interface per Hand und für den Bau von projektspezifischen UI-Werkzeugen hilfreich. Die Anforderungen und Bewertungskriterien geben Ihnen einen Überblick darüber, welche Faktoren in die Entwicklung einer Oberfläche einfließen.

Wenn Sie also ein konventionelles GUI Toolkit verwenden oder das UI mit projektspezifischen Methoden entwickeln, können Sie zum Beispiel anhand der oben aufgezeigten UI-Ausdehnung prüfen, ob und welche UI-Bereiche ihre Methoden und Werkzeuge abdecken.

Mit diesen Hilfsmitteln sind Sie in der Lage, Ihre UI-Entwicklungsmethodik auf Vollständigkeit und Robustheit zu prüfen und bei Bedarf zu vervollständigen.

Kapitel 4: Skizzieren

… das sich der Antwort auf die Frage widmet, wie man von Anforderungen zu einer Oberflächenbeschreibung kommt.

Kapitelziele

- *Wissen, wie man aus Anforderungen User Interface-Skizzen formt.*
- *Wissen, wie man den Umfang und die Organisation der Oberfläche bestimmt.*
- *Wissen, wie man den Gesamtumfang der Anwendung erfasst und wie man dabei die Überdetaillierung vermeidet.*

Das Sprichwort sagt: Aller Anfang ist schwer. Bei der Entwicklung einer Softwareanwendung trifft das Sprichwort in mehrfacher Hinsicht zu. Die Projektbeteiligten haben unterschiedliche Wissensstände und bringen unterschiedliche mentale Modelle der Anwendung mit.

In der Initialisierungsphase des Projekts setzt eine Flut von Informationen in Form von Anforderungen, Vorgängeranwendungen, Konzeptpapieren, Restriktionen sowie Erfahrungen der einzelnen Entwickler ein und will zu einem ersten geschlossenen Bild geformt werden.

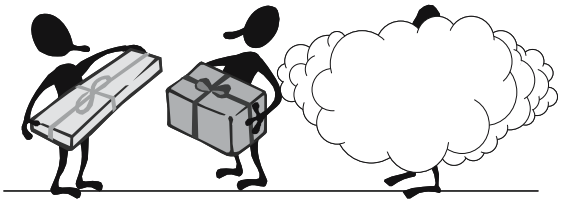


Bild 34: Projektbeteiligte bringen mentale Modelle der Anwendung mit

Die Projektbeteiligten bringen unterschiedliche mentale Modelle der zu entwickelnden Anwendung mit. Oft gehen die Spezialisten in den ersten Beschreibungen und Prototypen an manchen Stellen sehr ins Detail, während über andere Bereiche der Anwendung nur Schlagworte genannt werden.

Anwendungsbreite vor Funktionstiefe beschreiben

Die Herausforderung der ersten Projektphase liegt darin, einen Grundriss der Software zu formen, der die Breite der Anwendung vollständig wiedergibt, zugleich aber die Tiefe der Details begrenzt und nur so weit berücksichtigt, wie es für ein geschlossenes Bild notwendig ist.

Überdetaillieren vermeiden

Schwierig ist dabei, das Überdetaillieren der besonders interessanten oder wohlbekannten Bereiche der Anwendung zu vermeiden. Die Versuchung, im Rohbau der Software an einzelnen Stellen schon mal Parkett zu verlegen und einige Quadratmeter wohnlich einzurichten, ist groß. Umso enttäuschter ist man in der Regel, dass das Parkett wieder aufgerissen werden muss, wenn die Heizungsbauer kommen.

Logische Schritte identifizieren

Im Vordergrund der ersten User Interface-Skizzen stehen die ***logischen Schritte*** der Softwareanwendung sowie die Abgrenzung des Verwendungszwecks und der Inhalte dieser Schritte. Logische Schritte unterteilen das UI der Anwendung in Sequenzen aus in sich abgeschlossenen Tätigkeiten bzw. Aufgabenstellungen, die der Anwender im Dialog mit der Oberfläche durchführt.

Dabei ist es wichtiger, was die einzelnen logischen Schritte bewirken, also was am Ende eines Bildschirms oder einer Bildschirmfolge erreicht sein soll, als alle Felder der einzelnen Masken ausfindig zu machen. Oft genug stellt sich heraus, dass aus einer Bildschirmmaske zwei werden oder dass mehrere logische Schritte auf einer Bildschirmmaske vereinigt werden.

In Bezug auf die Oberfläche will man in der ersten Phase eines Softwareentwicklungsprojekts feststellen, wie viele und welche Bildschirmmasken die Anwendung in etwa hat, wie sich die Masken inhaltlich und bezüglich des Einsatzzwecks voneinander unterscheiden und wie sie zueinander in Beziehung stehen.

4.1 Methodenwahl

Für das Skizzieren und Detaillieren einer Oberfläche können verschiedene Beschreibungsformen verwendet werden.

Die nachfolgend aufgeführten, typischen Beschreibungsformen für Oberflächen sind unterschiedlich abstrakt und variieren von umgangssprachlich bis codierungsnah sowie von formlos bis algorithmisch.

- Umgangssprachliche UI-Beschreibung
- Tabellarische UI-Beschreibung
- Strukturierte UI-Beschreibung
- Formale UI-Modelle
- UI-Prototypen mit HTML
- Programmierte UI-Prototypen

Die Auswahl der Spezifikationsmethode kann zusammen mit der Wahl des UI-Entwicklungswerkzeugs oder unabhängig davon

erfolgen. Je formaler die gewählte UI-Beschreibungsmethode ist, desto mehr tritt allerdings die Frage der Unterstützung durch Werkzeuge in den Vordergrund.

Umgangssprachliche UI-Beschreibung

- Umgangssprache
- Händische Maskenentwürfe
- Benutzermodelle

Es gibt natürlich gute und schlechte beschreibende Spezifikationen. Die natürliche Sprache ist die für Menschen verständlichste, aber auch missverständlichste Form der Beschreibung einer Anwendungsoberfläche.

Erleben des Benutzers im Vordergrund

Oft haben umgangssprachliche Spezifikationen die Form von Erlebnisaufsätzen. Der Verfasser beschreibt dabei seine Erwartung an das Erleben der Anwendung beim Erledigen von Aufgaben.

Zu umgangssprachlichen Spezifikationen gehören ergänzend so genannte Benutzermodelle, bei denen ein Benutzer anhand von typischen Aufgaben und Verhaltensmustern beschrieben wird und so Anforderungen an Eigenschaften der Benutzerschnittstelle abgeleitet werden.

Andere umgangssprachliche Spezifikationen folgen einer vorgegebenen Gliederung, zum Beispiel:

- Zweck der Oberfläche
- Dialogfluss
- Dialoginhalte
- Verhalten
- Verwendung von Funktionen und Daten.

Bei Verfeinerung dieser Gliederung über mehrere Ebenen geht die umgangssprachliche Beschreibung in eine strukturierte UI-Beschreibung über.

Tabellarische UI-Beschreibung

- Tabellarische Beschreibung der Dialogseiten
- Maskenentwürfe mit Präsentationsprogramm

Die tabellenorientierte UI-Spezifikation, zum Beispiel mittels Excel und PowerPoint oder mit OpenOffice, wird in der Praxis oft angetroffen. Sie ist in der Regel weiterhin umgangssprachlich, definiert aber für einige Spezifikationsartefakte detailliert deren

Struktur. Diese Spezifikationsform wird oft beim Entwurf von einzelnen Dialogseiten angetroffen.

Kontrollelemente der Dialogseite im Vordergrund

Das Layout der Dialogseite wird zum Beispiel in PowerPoint erstellt. Die Liste der Elemente auf der Dialogseite wird mit Excel erstellt.

Mit dieser Spezifikationsform werden große Teile der Dialogperspektive abgedeckt. Auch Abläufe und Kontextbezüge können in tabellarischer Form spezifiziert werden. In der Praxis kommen tabellarische Spezifikationen, die über das Beschreiben einzelner Seiten hinausgehen, gegenüber z.B. Ablaufdiagrammen, vergleichsweise selten vor.

Strukturierte UI-Beschreibung

- Detaillierte hierarchische Struktur
- Datenhaltung innerhalb einer Repository-Struktur

Statische Struktur im Vordergrund

Diese Spezifikationsform gibt die Dokumentationsform, eine detaillierte Gliederung und formal strukturierende Regelungen für die Spezifikationsinhalte vor. Die Gliederung der Spezifikation wird feinkörnig herunter gebrochen und als Struktur in einem Repository bereitgestellt.

Die Struktur leitet den Spezifizierer beim Beschreiben der einzelnen UI-Eigenschaften an. Diese Form der UI-Spezifikation wird typischerweise mit Werkzeugen wie Rochade oder Doors erstellt.

Bei zunehmender Regelung der Strukturinhalte geht die strukturierte Beschreibung in ein formales UI-Modell über.

Formale UI-Modelle

- Formale Spezifikationssprache
- Verschiedene Sichten auf die Oberfläche

Eine formale Spezifikationssprache liefert eine Notation zur formalisierten, präzisen Beschreibung einer Softwareoberfläche.

Das UI wird in einem in sich konsistenten und geschlossenen Modell beschrieben. Mit Hilfe von Zusatzwerkzeugen kann die Einhaltung der Regeln der formalen Sprache und die Integrität des Modells geprüft werden.

UI-Konstruktion im Vordergrund

In der Regel muss der Spezifizierer die Philosophie des Modellierungswerkzeugs, eine mehr oder weniger aufwändige Modellierungsnotation und verschiedene werkzeugspezifische Kniffe erlernen. Für gewöhnlich ist die Lernkurve beim Einsatz formaler Modellierungsmethoden steil.

Die Akzeptanz des formalen Modellierens eines User Interface hängt ab von

- den Erfahrungen bzw. der Ausbildung der Personen mit Schlüsselrolle im Spezifikationsprozess,
- der Notation der UI-Beschreibungssprache,
- von der Einbindung der Entwickler in den Modellierungsprozess,
- und vom Komfort der Modellierungswerkzeuge.

UI-Prototypen mit HTML

- HTML Forms
- Layout mit HTML-Tabellen und -Layern

Die Spezifikation mit HTML Forms stützt sich auf die in HTML bzw. DHTML verfügbaren Möglichkeiten zum Definieren und zum Anordnen von Kontrollelementen auf Masken.

UI-Darstellung im Vordergrund

Diese Spezifikationsform erlaubt das detaillierte Festlegen der Dialogabfolge und der Dialoginhalte sowie Interaktionen.

Sie liefert aber für sich genommen keine vollständige UI-Spezifikation, sondern ergänzt andere Spezifikationsformen mit techniknahen Maskenentwürfen in HTML, z.B. eine strukturierte UI-Beschreibung.

Programmierte UI-Prototypen

- Ausführbare Oberfläche
- Codiert in einer Programmiersprache
- Verwendet in der Regel ein GUI Toolkit

Die Oberfläche der Anwendung wird durch ablauffähige programmierte Beispiele definiert. Die Eigenschaften der Oberfläche sind nicht explizit beschrieben, sondern liegen implizit in den erfahrbaren Eigenschaften des programmierten UI-Prototypen.

UI-Implementierung im Vordergrund

Der Prototyp liefert die für den Anwendervertreter intuitivste Form des Erlebens des Oberflächenentwurfs. Allerdings: Den Oberflächenentwurf verfeinern und detaillieren können der Anwendervertreter und der Spezifizierer nur zusammen mit einem Programmierer.

UI Design-Werkzeuge bieten in der Regel umfangreiche Unterstützung für das Screen Design. Oft stellen sie Bibliotheken von Dialogbausteinen und Kontrollelementen, die bereits für die Zielplattform zugeschnitten sind, zur Verfügung. Die unterstützten UI-Sichten sind Dialogperspektive und Strukturperspektive.

Die Codierung des Oberflächenentwurfs in einer Programmiersprache bietet den maximalen Grad an Flexibilität, unterstützt jedoch kein Anforderungsmanagement, keine modellgetriebene Vorgehensweise, keine Technologiekapselung, keine Wiederverwendung und keine Aufteilung der UI-Beschreibung in UI-Perspektiven.

Kriterien der Methodenwahl

Der Werdegang einer Oberfläche beginnt mit ersten Anforderungen. Der Weg des Entwurfs führt von ersten Skizzen über ausführbare Prototypen bis zu einer implementierungsreifen Detailspezifikation.

Bei der Wahl von Methoden und Werkzeugen sollten Sie darauf achten, dass diese einander ergänzen und eine stufenweise Verfeinerung der UI-Definition unterstützen.

Bei der Methodenwahl für die Entwicklung von User Interfaces steht dieses Ziel im Vordergrund: Eine Modellierungsumgebung zu haben, mit der die UI-Beschreibung von groben Skizzen schrittweise zu einer exakten Spezifikation verfeinert werden kann. Die Modellierungsumgebung sollte von intuitiven Spezifikationsformen (zusammenklickbare Masken, Listen von Maskenelementen) bis hin zu detaillierten formalen Beschreibungen (z.B. maschinenlesbare XML-Spezifikation) skalierbar sein.

Akzeptanzkriterien einer Methode zur UI-Entwicklung

Folgende Faktoren begünstigen die Akzeptanz einer Entwicklungsmethode für User Interfaces:

- Einfach erlernbare Syntax und Semantik der Modelle
- Einfache grafische bzw. tabellarische Grundnotation
- Modellierbarkeit in allgemein gebräuchlichen Büroanwendungen (im besten Fall etwa Powerpoint oder Excel mit einem Plugin, der Editierhilfe und Modellvalidierung bietet)
- Fachleute modellieren „in Breite“ (Use Case), Entwickler „in Tiefe“ (Detailablauf)
- Abschöpfbarkeit / Destillierbarkeit (d.h. Herausziehen des Informationsgehalts) des UI-Modells aus Common Tools (z.B. Office-Programme) in ein formales Modell (UML, XML)
- Vertiefende Modellierungswerkzeuge mit einer Detailnotation auf UML- oder XML-Basis
- Möglichkeit des Round Trip Modelling (hin und her Wechsel zwischen Grundform und Detailform des Modells ohne Verlust von Informationen)

- Umsetzung von WYSIWYG (What You See Is What You Get) Ansätzen in der Methode

4.2 Systemgrenzen abstecken

Finden der Eingangsdaten, der Ergebnisse und der Hauptschritte der Anwendung

Für einen ersten Überblick über die geplante Softwareanwendung ist ein Diagramm hilfreich, das die Grenzen des Systems absteckt.

Das Diagramm zeigt die Anwendersicht auf die Hauptschritte der Anwendung und formt einen Bearbeitungsfluss von den Eingangsinformationen zu den Zielergebnissen der Software.

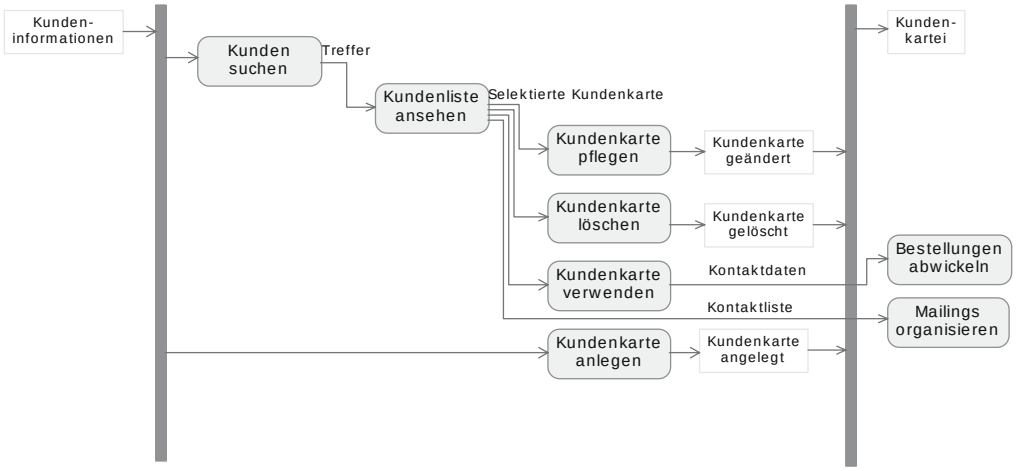


Bild 35: Systemgrenzen am Beispiel einer Kundenverwaltung

Die abgerundeten Vierecke zeigen die oberste Ebene der Anwenderaktivitäten. Die von links in die Aktivitäten mündenden Pfeile repräsentieren Eingangsinformationen. Die rechts aus den Aktivitäten herausgehenden Pfeile repräsentieren Ergebnisse.

Die durch einen Pfeil transportierten Informationen kann man direkt an den Pfeil schreiben oder in einen rechteckigen Kasten einfassen.

Ein Pfeil, der in der obigen Abbildung zwei Aktivitäten miteinander verbindet, bedeutet, dass das Ergebnis einer Aktivität in einer anderen Aktivität als Eingangsgröße verwendet wird.

Die senkrechten Balken zeigen die Grenzen der Anwendung an.

IDEF0 mit UML-Aktivitätsdiagrammen

Die IDEF0-Diagramme sind eine Weiterentwicklung von SADT (Structured Analysis and Design Technique) und wurden ur-

sprünglich unter anderem von der NASA zur Beschreibung von Prozessen verwendet [Feldmann 1998].

UML-Aktivitätsdiagramme haben eine den IDEF0-Diagrammen ähnliche Semantik. In der UI-orientierten Architektur verwende ich eine von mir auf Aktivitätsdiagramme adaptierte Form von IDEF0-Diagrammen. Das Diagramm in der obigen Abbildung ist ein UML-Aktivitätsdiagramm, dessen Aufbau und Semantik an einem IDEF0-Diagramm orientiert ist.

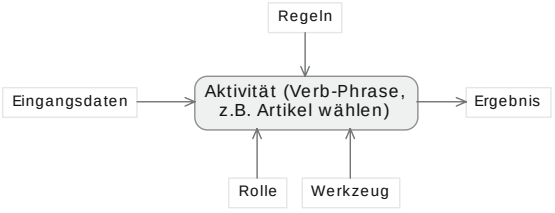
Weg zum Ergebnis der Anwendung im Vordergrund

Ein IDEF0-Diagramm ist eine Prozessvorschrift, bei der nicht der Ablauf, sondern der Zusammenhang zwischen den Voraussetzungen und Ergebnissen von Tätigkeiten und den für die Tätigkeiten benötigten Mitteln und Regeln im Vordergrund steht.

Die im Buch über UI-orientierte Softwarearchitektur verwendeten IDEF0-ähnlichen Diagramme bedienen sich der Semantik von Aktivitätsdiagrammen mit einigen an IDEF0 angelehnten Spezialisierungen der Übergänge zwischen Aktionszuständen.

Die nachfolgende Übersicht zeigt, wie IDEF0-Diagramme in der UI-orientierten Softwarearchitektur auf UML-Aktivitätsdiagramme abgebildet werden.

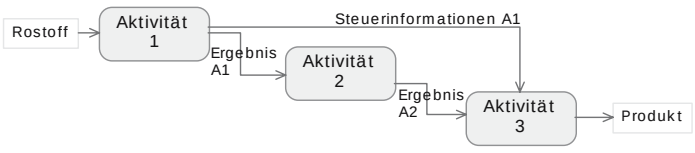
Bild 36: IDEF0 Activity Box



Die Activity Box steht für eine Aktivität des Anwenders. Die Aktivität wird durch eine Verb Phrase, also durch ein Sachwort und ein Tätigkeitswort - z.B. „Artikel wählen“ - ausgedrückt.

- Von links werden in die Aktivität die Eingangsinformationen eingespeist.
- Von oben und unten wird die Aktivität mit Regeln, durchführenden Personen (Rolle) und Mechanismen (Werkzeuge, Systeme) versorgt.
- Nach rechts gehen aus der Aktivität Ergebnisse hervor.

Bild 37: Aktivitäten verketten



Die Datenflüsse verketten Aktivitäten zu Prozessen. Das Ergebnis einer Aktivität kann als Eingangsinformation oder als Regel für andere Aktivitäten dienen. Man kann die Eingangsgröße der ersten Aktivität als Rohstoff, den die Software verarbeitet, und die Ausgangsgröße der letzten Aktivität als Produkt, das die Anwendung der Software liefert, sehen.

Nach den Regeln von IDEF0 sollte ein Diagramm aus mindestens drei, höchstens aber aus sechs Aktivitäten bestehen. Für die einzelnen Aktivitäten des Diagramms wird auf der nächsten Detaillierungsstufe jeweils ein eigenes Diagramm erstellt.

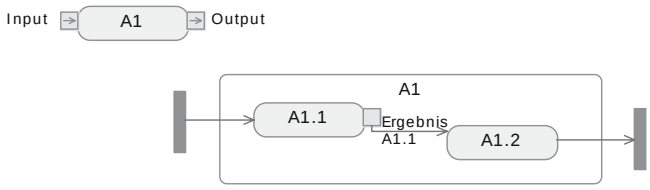


Bild 38: Dekomposition und Border Lines

Border Lines kennzeichnen Eingangs-, Ausgangs- und Regelgrößen, welche die Grenzen des Diagramms (und beim Diagramm der obersten Abstraktionsstufe die Systemgrenzen) überschreiten.

IDEF0 in UML

Gehen Sie wie folgt vor, um IDEF0-ähnliche Aktivitätsdiagramme für Ihre Anwendung zu erstellen:

- Die IDEF0 Activity Box wird im UML-Aktivitätsdiagramm durch einen Aktionszustand abgebildet.
- Eingangsdaten, Ergebnisse, Regeln und Mechanismen (Rollen, Werkzeuge, Systeme) aus IDEF0 werden im UML-Aktivitätsdiagramm durch Übergänge sowie durch Eingabe- und Ausgabe-Pins abgebildet.
- Übergänge und Eingabe-Pins, die von links in Aktionszustände münden, repräsentieren Eingangsdaten.
- Übergänge und Ausgabe-Pins, die rechts aus Aktionszuständen herauskommen, repräsentieren Ergebnisse.
- Übergänge, die von oben in Aktionszustände münden, repräsentieren Regeln und Restriktionen.
- Übergänge, die von unten in Aktionszustände münden, repräsentieren Mechanismen.
- Die von den Übergängen transportierten Informationen werden wahlweise direkt an den Übergangskanten annotiert oder durch Objektflusszustände repräsentiert.
- Synchronisationsleisten repräsentieren IDEF0 Border Lines.

4.3 Ablaufstruktur formen

Mit ***Ablaufstruktur*** ist eine Aufteilung der Anwendung in Phasen, Tätigkeiten und einzelne Arbeitsschritte gemeint. Diese Aufteilung erfolgt aus Anwendersicht: Sie liefert einen Überblick darüber, wie das, was der Anwender mit der Anwendung tun kann „von Anfang bis Ende" aneinandergereiht und „vom Groben ins Feine" untergliedert werden kann.

Die Ablaufstruktur beschreibt keine Ablaufdynamik

Die vielen verschiedenen ***Fallabhängigkeiten*** in den Abläufen einer Software bleiben beim Aufstellen der Ablaufstruktur zunächst außen vor.

Die ***Ablaufstruktur*** ist sozusagen ein statisches Gerüst für die ***Ablaufdynamik***, die aber erst bei der späteren Detaillierung folgt.

Die Ablaufstruktur einer Softwareoberfläche beinhaltet folgende Informationen:

- Ablaufgliederung (Hierarchie der logischen Schritte),
- Eigenschaften der in der Ablaufgliederung aufgenommenen Schritten (zum Beispiel Sequence Steps und Service Steps),
- Enter- und Exit-Behandlung (Aktionen beim Betreten und Verlassen) logischer Schritte.

Die Ablaufgliederung einer Anwendungsoberfläche wird durch die Hierarchie der ***logischen Schritte*** dargestellt.

Als Näherungswert können Sie sich einen logischen Schritt zunächst als einen Dialogbildschirm einer grafischen Oberfläche vorstellen (siehe auch Kapitel „Oberfläche"). Logische Schritte sind eine Detaillierung von Use Cases. Bei der Beschreibung von User Interfaces bieten logische Schritte gegenüber Use Cases folgende Vorteile:

- Logische Schritte können hierarchisch gegliedert sein
- Logische Schritte drücken eine Ausführungsreihenfolge aus
- Man kann Haupt- und Nebenpfade im Ablauf der Anwendung unterscheiden
- Man kann verschiedene Typen von Schritten unterscheiden

Kanonische Sequenzen bilden

Man kann jede Anwendung in ***kanonische Sequenzen*** aus logischen Schritten gliedern. Das heißt: Man bringt alles, was

man mit der Anwendung machen kann, in eine nach Tätigkeiten und Arbeitsschritten gegliederte Grundreihenfolge.

Elementare und zusammengesetzte Schritte

Die Anwendungsoberfläche wird in einen Baum aus logischen Zuständen gegliedert. Diese Zustandsknoten spiegeln logische Schritte (Arbeitsschritte, Tätigkeiten, Phasen) der Anwendungsoberfläche wider.

Die Blätter des Baums repräsentieren ***elementare*** (nicht weiter unterteilte) logische Arbeitsschritte. Die hierarchisch über den elementaren Arbeitsschritten liegenden Knoten repräsentieren ***zusammengesetzte*** Schritte, z.B. Tätigkeiten und Phasen.

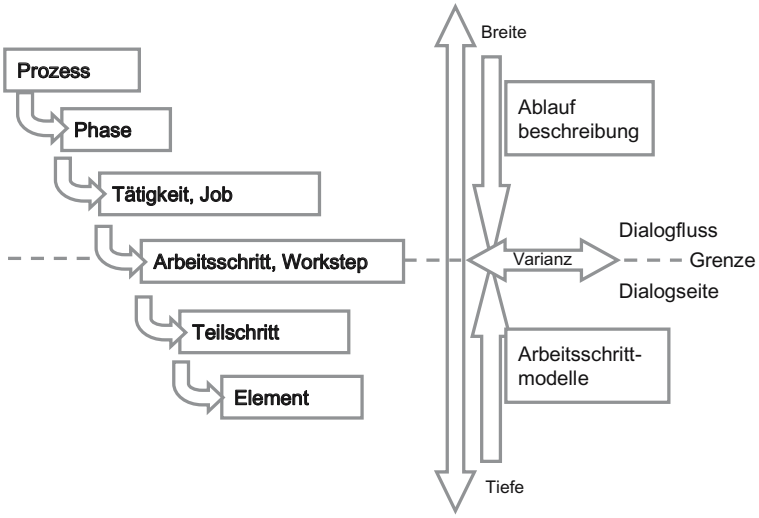


Bild 39: Prozessgliederung, Dialogfluss und Dialogseiten

Das obige Bild zeigt die Hierarchie aus Arbeitsschritten, Tätigkeiten und Phasen und die Verankerung der Ablaufsteuerung auf der Ebene eines Dialog-Arbeitsschritts (also einer Dialogseite).

Das Verankern auf Arbeitsschrittebene bedeutet, dass man beim Spezifizieren nach Ablaufschritten sucht, die eine in sich so weit abgeschlossene Einheit bilden, dass sie auf einer Dialogseite abgehandelt wird.

Die Dialogseite ist für den Arbeitsschritt ein Kommunikationsmedium, mit dem die zum Arbeitsschritt gehörenden Daten dem Anwender angezeigt werden, Eingabedaten erfasst und die Befehle zum Fortsetzen des Workflows empfangen werden.

Ein Arbeitsschritt sollte ein definierbares Ergebnis haben, so dass man z.B. den Workflow nach einer Unterbrechung des Arbeitens mit der Anwendung auf dem gleichen Arbeitsschritt wieder aufsetzen kann.

Ein Teilschritt ist ein Teil eines Arbeitsschritts, das aber im Workflow kein in sich eigenständiges Ergebnis hat. Auf einer Dialogseite, die einen Arbeitsschritt abbildet, kann der Teilschritt durch

einen Dialogabschnitt abgebildet werden. Ein Dialogabschnitt wird auf der Dialogseite z.B. durch eine Group-Box, eine Abschnitttrennlinie oder durch einen Karteireiter (bei einem Wizard-Dialog) repräsentiert.

Beispiel: Ablaufstruktur der Kundenverwaltung

Erfassen und Strukturieren der logischen Schritte

Die folgende Gliederung zeigt die Struktur der logischen Schritte für das Beispiel der Kundenverwaltung.

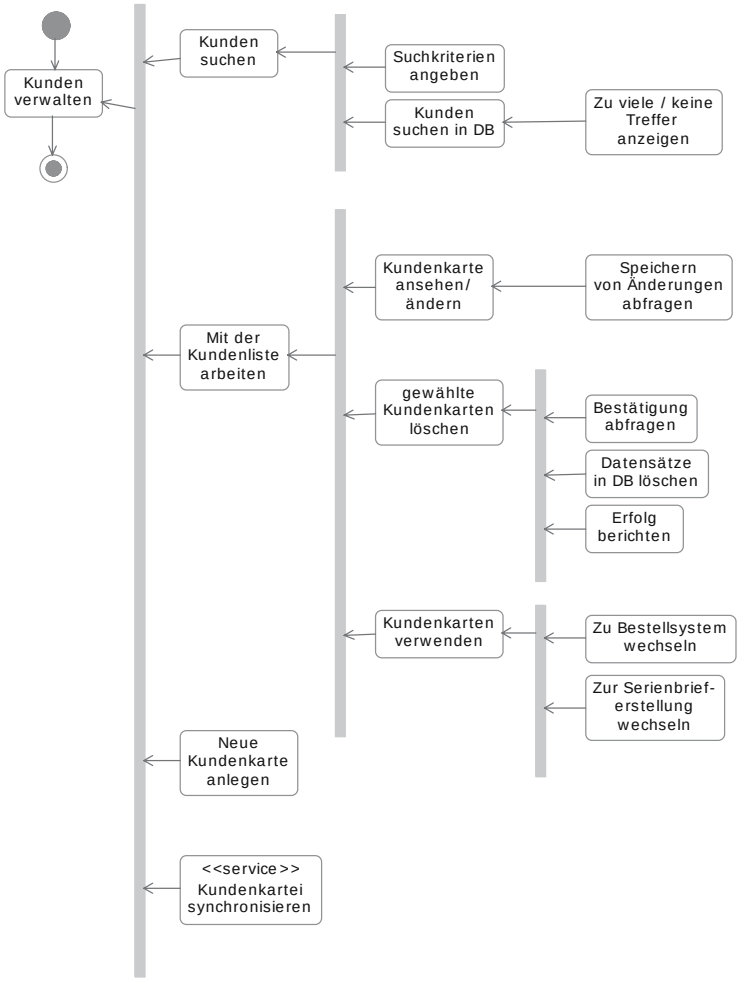


Bild 40: Struktur der logischen Schritte in der Kundenverwaltung

Durch die Darstellung der Anwendungsoberfläche als eine Hierarchie von logischen Schritten kann man ihre Dialogentitäten (z.B. GUI Screens oder Sprachmenüs) identifizieren und gliedern.

4.4 Logische Schritte modellieren

Das Konzept des logischen Schritts spielt für die Erklärung der Abläufe in einer Softwareoberfläche eine wichtige Rolle.

Logische Schritte sind Stationen im Ablauf der Arbeit des Anwenders mit der Oberfläche. Sie sind ***Bausteine der Ablaufstruktur*** der Anwendung.

Unterschiedliche Schritte

Nicht alle logischen Schritte sind gleich. Sie können sich in der Art der Aufeinanderfolge, in Repräsentation an der Oberfläche und in anderen Eigenschaften unterscheiden.

Ausführungsmodus: Sequence-Step und Service-Step

Es gibt Schritte, die hintereinander ausgeführt werden und so den Bearbeitungsfluss der Anwendung bilden. Sie heißen ***Sequence-Steps***. Andere Schritte werden wie ein Dienst verwendet. Nach ihrem Durchführen wird der Bearbeitungsfluss anschließend an der Stelle fortgesetzt, von der aus der Dienst abgerufen wurde. Diese Schritte heißen ***Service-Steps***. Mit Hilfe von Service-Steps lassen sich Unterbrechungen im Bearbeitungsfluss, z.B. modale Meldungen und Popup-Dialoge modellieren.

Modalität: Machine-Step und Dialog-Step

Logische Schritte können unterschiedliche Modalitäten haben, also mit grafischer Ein- und Ausgabe, Ein- und Ausgabe mittels gesprochener Sprache oder anderen Rezeptoren und Manifestoren ausgestattet sein.

Es gibt Schritte, die ohne Benutzerinteraktion auskommen (***Machine-Steps***) und solche, die mit einem Dialogbildschirm und/oder mit einem Sprachdialog ausgestattet sind. Diese Schritte heißen ***Dialog-Steps***.

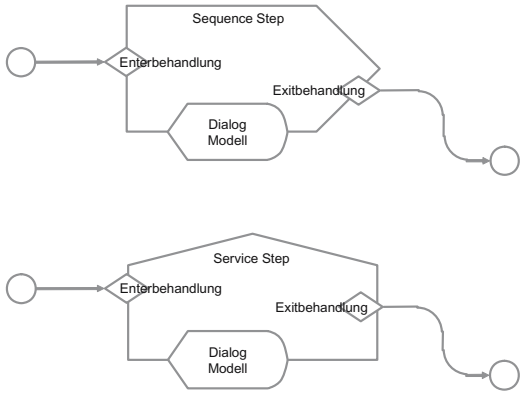


Bild 41: Grafische Darstellung für logische Schritte

Die obige Abbildung zeigt einen Sequence Step (Kasten mit Spitze nach rechts) und einen Service Step (Kasten mit Spitze nach oben). Die Rauten am linken und rechten Rand der Step-

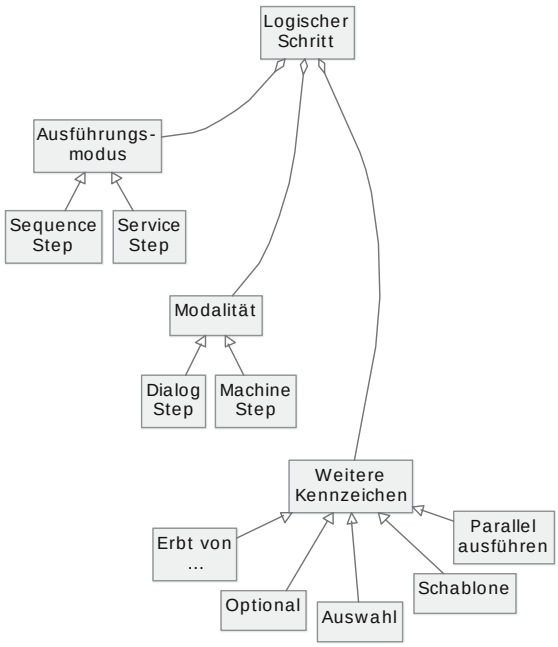
Symbole repräsentieren die Enter- und die Exitbehandlung der Steps. Beide Steps sind Dialog Steps. Das wird durch das Dialogsymbol am unteren Rand des Step-Symbols angezeigt.

Man kann diese Symbole beim Modellieren von Ablaufstrukturen verwenden. Sie helfen dabei, sich einen schnellen Überblick über den logischen Aufbau der Anwendung zu verschaffen und auf einen Blick Dialogschritte, Sequenzen und Services zu identifizieren.

Ein logischer Schritt hat verschiedene Kennzeichen, die miteinander kombiniert werden können. Die Kennzeichen bestimmen die Eigenschaften des Schritts in der modellierten Oberfläche.

Das folgende Diagramm zeigt einen Überblick über die Kennzeichen für logische Schritte.

Bild 42: Wichtigste Kennzeichen für logische Schritte



Die Eigenschaft ***Ausführungsmodus*** kennzeichnet, wie sich der Schritt im Kontrollfluss der Anwendung verhält. Die beiden Ausprägungen des Ausführungsmodus sind ***Sequence-Step*** und ***Service-Step***. Die Defaultausprägung von Ausführungsmodus ist Sequence-Step.

Logische Schritte als Service ausführen

Falls Ausführungsmodus = Service-Step angegeben wurde, wird der Schritt als Dienst ausgeführt und sein Standardnachfolger ist der aufrufende Schritt. Das heißt: Der laufende Arbeitsschritt wird unterbrochen, um einen Dienst auszuführen (z.B. eine wichtige Meldung in einem Popup-Dialog zu lesen und bestäti-

gen). Anschließend wird der unterbrochene Arbeitsschritt in der Regel fortgesetzt.

Logische Schritte als Sequenz ausführen

Wurde Ausführungsmodus = Sequence-Step oder kein Ausführungsmodus angegeben, dann wird der Schritt als Sequenzbaustein ausgeführt. Sein Standardnachfolger ist dann der nächste Schritt der gleichen Gliederungsstufe. Damit bilden alle Geschwister-Knoten (sibling nodes) im Gliederungsbaum eine Sequenz in Definitionsreihenfolge. Das heißt: Nacheinander auf der gleichen Gliederungsstufe definierte Schritte werden als eine logische Abfolge betrachtet.

Der Unterschied zwischen Service-Step und Sequence-Step ist also, dass üblicherweise nach Ausführung eines Service-Steps zum aufrufenden Schritt zurückgekehrt wird, nach Ausführung eines Sequence-Steps dagegen mit dem Nachfolger-Schritt in Lesereihenfolge fortgefahren wird.

Services mit innen liegenden Sequenzen

Ein Service-Step kann im Übrigen auf der nächsten Gliederungsebene wiederum in Sequence-Steps unterteilt werden. Das ist zum Beispiel dann sinnvoll, wenn größere Abläufe (z.B. die Erstanmeldung bei einem Web-Portal) als Service realisiert werden und der Anwender anschließend da weitermachen kann, wo er gerade war, bevor die Erstanmeldung in seinen Workflow „eingeschoben“ wurde.

Dadurch, dass man einem Schritt einen Ausführungsmodus zuweist, bestimmt man den Unterschied zwischen workfloworientierten und dienstorientierten Teilen der Oberfläche. Die workfloworientierten Teile stellen die Standardabläufe der Oberfläche dar. Die dienstorientierten Teile stellen die bei Bedarf abrufbaren Funktionen, welche Standardabläufe unterbrechen und nach Ausführung zurückkehren, dar.

Die Auswirkung auf die Abläufe in der Anwendungsoberfläche wird beim Vergleich mit herkömmlichen Programmiersprachen deutlich. Durch die Einführung des Flussmodus wird der Unterschied zwischen absolutem Sprung im Ablauf und einem Unterablaufaufruf nicht mehr vom Aufrufer, sondern von Aufgerufenen bestimmt.

Der Vorteil dieser Neuerung liegt darin, dass der aufrufende Step nicht unterscheiden muss, ob der angeforderte Arbeitsschritt Bestandteil des Standardablaufs oder eine Ausnahme bzw. ein Dienst ist.

Die Eigenschaft ***Modalität*** kennzeichnet den bevorzugten Kommunikationskanal des Steps, d.h. ob und auf welchem Weg [Grafischer Dialog, Sprachkommandos, etc.] der Step hauptsächlich mit dem Anwender verkehrt.

Die Modalität hat klassifizierenden Charakter und wird vor allem zu Validierungs- und Abfragezwecken auf Ebene der UI-Beschreibung verwendet. Eine einschränkende Auswirkung auf die zu einem Step gehörenden Dialogelemente und auf deren Ausführung hat die Modalität nicht. In einem Dialog können zum Beispiel Kontrollelemente für Sprachkommandos und Mausbedienung kombiniert sein.

Falls keine Modalität angegeben wurde, hat der Step die neutrale Modalität LogicalStep, was bedeutet, dass keine Modalität bevorzugt wird. Weitere Modalitäten sind DialogStep, GuiStep, VoiceStep und MachineStep.

Die im Diagramm angedeuteten ***weiteren Kennzeichen*** zeigen z.B. das Erben von Eigenschaften von anderen Steps oder die Verwendung eines Steps als Schablone an. Ebenso kann es notwendig sein, parallel auszuführende Schritte zu kennzeichnen, oder anzuzeigen, dass aus einer Gruppe von Schritten nur einer ausgeführt werden soll oder dass ein Schritt optional ist.

Aktive, passive und unbeteiligte Schritte

Zu jedem Zeitpunkt des Oberflächenlaufs ist genau ein Blatt im Baum der logischen Schritte aktiv.

In dem gerade aktiven logischen Schritt wartet die Oberfläche auf Ereignisse und reagiert auf diese aus dem Blickwinkel dieses aktiven Schritts.

Obwohl zu jedem Zeitpunkt nur ein Schritt aktiv ist, gibt es Schritte, die zugleich mit dem aktiven Schritt ***semi-aktiv*** bleiben. Diese semi-aktiven Schritte kommen dadurch zustande, dass Schritte hierarchisch gegliedert sind und weil logische Schritte als Dienst im Auftrag von anderen Schritten ausgeführt werden können.

Die in der Ablaufhierarchie übergeordneten Schritte (ich nenne sie ***Lead-Steps***) bleiben semi-aktiv (auf Reaktivierung wartend), weil der untergeordnete Schritt (z.B. ein Service-Step) im Auftrag des übergeordneten Schritts durchgeführt wird. Der Lead-Step wird nach dem Ende des untergeordneten Schritts reaktiviert.

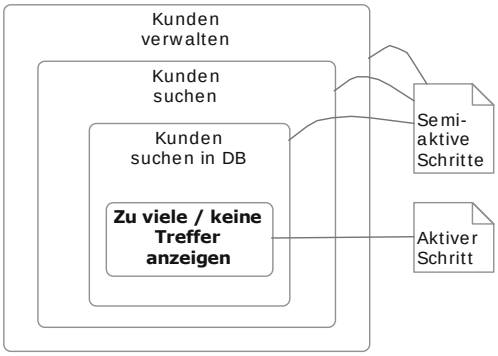


Bild 43: Aktiver Schritt und hierarchisch semi-aktive Schritte

Im obigen Beispiel sind dem aktiven Schritt „Zu viele / keine Treffer anzeigen“ die Schritte „Kunden suchen in DB“, „Kunden suchen“ und „Kunden verwalten“ hierarchisch übergeordnet und daher semi-aktiv.

Man kann sich die gesamte Anwendungsoberfläche als einen ***einzelnen*** logischen Schritt „UI ausführen“ vorstellen. Dieser ***oberste*** Schritt gliedert sich in eine Hierarchie von ***untergeordneten*** Schritten.

Im Beispiel heißt das: Wenn der hierarchisch oberste logische Schritt „Kunden verwalten“ zu Ende geht, wird damit auch die Anwendungsoberfläche beendet.

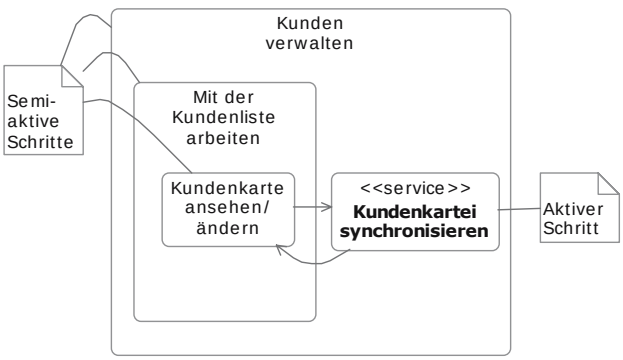


Bild 44: Aktiver Schritt und orthogonal semi-aktive Schritte

Das zweite Beispiel zeigt einen Service-Step „Kundenkartei synchronisieren“, der aus einem nicht hierarchisch übergeordneten Schritt heraus aktiviert wird. Auch hier gilt: Der aufrufende Schritt bleibt semi-aktiv und wird nach dem Ende des Service-Steps reaktiviert.

Aus Sicht der Laufzeit kann man die logischen Schritte einer Anwendungsoberfläche also zusammenfassend in drei Gruppen unterteilen.

- Aktiver Schritt
- Passive (oder semi-aktive) Schritte
- Unbeteiligte Schritte

Der aktive Schritt ist der Schritt, der gerade ausgeführt wird. Die passiven (oder semi-aktiven) Schritte sind alle Schritte, die vor dem aktiven Schritt begonnen, aber nicht beendet wurden.

Alle anderen in der Ablaufstruktur definierten Schritte sind „unbeteiligt", spielen also zu diesem Zeitpunkt der Laufzeit weder eine aktive, noch eine passive Rolle.

Betreten und Verlassen logischer Schritte

Die Aktionen beim Betreten eines Schritts werden als „Enter-Behandlung" bezeichnet. Analog dazu heißen die Aktionen beim Verlassen des Schritts „Exit-Behandlung".

Bei Ausführen eines zusammengesetzten Schritts (siehe oben) werden also die Startaktionen (Enter-Behandlung) durchgeführt und dann der erste untergeordnete Schritt aktiviert. Diese Prozedur wird wiederholt, bis ein elementarer (also nicht zusammengesetzter) Schritt erreicht ist. In einem elementaren Schritt folgt auf die Enter-Behandlung das Durchführen von Aufgaben (z.B. das Ausführen einer Dialogseite) und zum Schluss die Exit-Behandlung. Nach dem Ende der Exit-Behandlung wird die Kontrolle an den hierarchisch übergeordneten Schritt zurückgegeben.

Man kann sich die Kontrolle dieser Aktivitätshierarchie wie einen Stapel aus begonnenen Schritten vorstellen. Auf dem unteren Ende des Stapels liegt der hierarchisch höchste (allen anderen übergeordnete) Schritt „UI der Anwendung ausführen"; auf dem oberen Ende des Stapels befindet sich der aktive Schritt. Alle unter dem aktiven Schritt auf dem Stapel liegenden Schritte sind passiv oder semi-aktiv, d.h. sie warten auf ihre Reaktivierung nach dem Ende der weiter oben auf dem Stapel liegenden Schritte.

Mit dem vollständigen Abbauen des Aktivitätsstapels wird der hierarchisch höchste Step „UI der Anwendung ausführen" beendet und damit der Lauf der Anwendungsoberfläche terminiert.

4.5 Dialoginhalte abgrenzen

Ziel des Abgrenzens von Dialoginhalten ist es, für die einzelnen logischen Schritte der Anwendung zu wissen, welche Informationen auf die dazugehörige Dialogseite gehören.

Zum Abgrenzen gehört es zu ermitteln, welche Informationen auf einer Dialogseite zusammengefasst werden sollen und wel-

che Informationen unter Umständen auf zwei Dialogseiten aufgeteilt werden müssen.

Das Ergebnis dieser Abgrenzung ist ein stabiles Bild davon, welche Dialoginhalte hinter einzelnen logischen Schritten stehen. Durch das Abgrenzen der Inhalte der Dialogseiten wird die Ablaufstruktur abgesichert und der Datenumfang der logischen Schritte aneinander angeglichen.

4.6 Nutzen der UI-Skizzen

Der Nutzen der UI-Modelle in der Startphase ist, dass Kunden und Entwickler einen vollständigen Überblick über Umfang und Struktur der Anwendungsoberfläche gewinnen. Der Überblick erfolgt in der Breite, die Tiefe folgt später und wird zunächst vermieden. Das erste UI-Modell wird von Bedienungsprototypen, Designs und Schaubildern einzelner Dialogseiten der Anwendung begleitet.

Die ersten UI-Modelle ermöglichen den Projektbeteiligten die Beurteilung des Umfangs der Anwendung, der Vollständigkeit der Kernanforderungen und der Arbeitsweise der Oberfläche (Maskenfluss, Aufteilung, Dialogführung). Auf der Basis dieses Inception-Modells können erste Klassenmodelle und vertikale Durchstiche im Kontext der Anwendungsbreite und der Systemgrenzen erstellt werden.

Die Kenntnis des Kontextes senkt das Risiko, dass Anforderungen an die Funktionalität hinsichtlich ihrer Nutzung falsch eingeschätzt werden und wesentliche Anforderungsteile übersehen werden. Ebenso wird durch den Kontext vermieden, in Funktionen Anforderungen hineinzuinterpretieren, die sich aus der Verwendung an der Oberfläche nicht ergeben.

UI-Überblicksmodelle schaffen ein geschlossenes Bild der Anwendung und tragen so entscheidend dazu bei, dass von Anfang an ein gemeinsames Verständnis der Anforderungen und des Aufbaus der Software geformt wird.

Im Kapitel Skizzieren habe ich Ihnen eine Vorgehensweise für das Skizzieren einer Anwendungsoberfläche vorgestellt. Das folgende Diagramm fasst die vorgestellten Tätigkeiten und den Arbeitsfluss beim Skizzieren einer Anwendungsoberfläche zusammen.

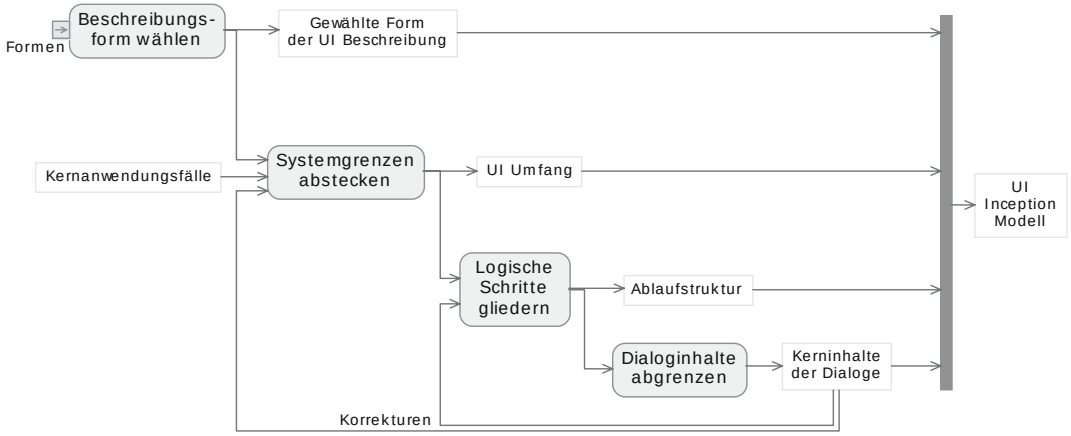


Bild 45: Vorgehenskarte für das Skizzieren der Oberfläche

Das obige Vorgehensschema leitet Sie in der ersten Phase des Anwendungs- und UI-Entwurfs an. Ich bezeichne diese Iteration der UI-Entwicklung in Anlehnung an RUP als ***UI-Inception*** (Konzeptualisierung der Softwareoberfläche). Das Ergebnis der UI-Inception ist ein Modell, das als Grundlage für alle weiteren Detaillierungen der UI-Beschreibung dient. Ich empfehle Ihnen daher, die Tätigkeiten der Konzeptualisierung gründlich durchzuführen, um sicherzustellen, dass Sie beim späteren Detaillieren des UI-Entwurfs auf einen stabilen UI-Umfang, auf eine robuste Ablaufstruktur und auf zuverlässige Dialoginhalte zurückgreifen können.

Die Merkmale einer soliden UI-Inception können nach folgenden Kriterien bewertet werden:

- ***Stabiler UI-Umfang:***
  - Klare Systemgrenzen (siehe Abschnitt „Systemgrenzen abstecken“)
  - Abdeckung der Anwendungsfälle aller Anwenderrollen
- ***Robuste Ablaufstruktur:***
  - Gleichmäßige Gliederungstiefe
  - Keine überflüssigen Gliederungsebenen
  - Ablaufschritte geben in sich abgeschlossene Anwendertätigkeiten wieder
- ***Zuverlässige Dialoginhalte:***
  - Dialogschritte haben ein definiertes Ergebnis
  - Es ist definiert, was pro Dialogseite eingegeben werden muss und was der Anwender sehen will (Kerninhalte)

Mit einer soliden UI-Skizze in Form eines Inception-Modells vermeiden Sie, dass bereits erstellte Teile des UI-Entwurfs später verworfen werden müssen.

Sie verhindern damit auch weiße Flecken auf der Landkarte der Anwendung. Das spätere Entdecken solcher Lücken kann unerwartete zusätzliche Aufwände und Terminengpässe durch Wiederholen von bereits erledigt geglaubten Entwurfsschritten zur Folge haben.

Kapitel 5: Detaillieren

… das den Weg von einem groben User Interface-Gerüst zu einer umsetzungsreifen Spezifikation der Anwendungsoberfläche aufzeigt.

Kapitelziele

- *Wissen, wie man eine Oberflächenbeschreibung in Iterationen detailliert.*
- *Wissen, wie man eine gleichmäßige Detailtiefe erreicht und behält.*
- *Wissen, wie man Stand und Qualität der Oberflächenbeschreibung messbar macht.*

Besitzt man als Konstrukteur einer neuen Maschine einmal eine genaue Vorstellung davon, wie etwas im Detail funktionieren wird, dann kann man es in einem Zug formen. Selten bis nie stimmt aber die erste Skizze mit dem fertig entwickelten Produkt überein. Beim Entwickeln muss man sich oft an die Lösung herantasten und erreicht die Lösung schrittweise; es handelt sich eben um eine „Ent-Wicklung“ im wörtlichen Sinn.

Benutzeroberflächen werden nicht in einem Aufwasch, sondern schrittweise entwickelt. Für User Interfaces ist das schrittweise Entwickeln der richtigen Lösung von vielleicht noch größerer Bedeutung als für andere Konstruktionen: Man muss fühlen, wie die Oberfläche „in der Hand liegt“ und „sitzt“, um Verbesserungspotentiale und richtige Lösungen zu erkennen.

Bild 46: User Interfaces werden schrittweise entwickelt

Die Hauptkritik an bestehenden GUI Toolkits ist: Sie setzen voraus, dass der Anwendervertreter / Spezifizierer / Programmierer bereits ein fertiges, detailliertes, weitgehend fehlerfreies und stabiles Bild der Oberfläche im Kopf mitbringt.

Bestehende Werkzeuge setzen ein weitgehend feststehendes Konzept voraus.

Die gängigen UI-Werkzeuge bieten für die iterative und evolutionäre Entwicklung von Oberflächen wenig Unterstützung. Die Werkzeuge unterstützen den Entwickler in erster Linie dabei, seine bereits weitgehend feststehende Lösung in die zur Verfügung stehenden Formalismen zu übertragen.

Viele GUI-Entwicklungswerkzeuge fordern vom Anwender von Anfang an die Festlegung von Details, die möglicherweise erst später geklärt werden können. Die erstellten GUI-Modelle erweisen sich als zäh, wenn der Entwickler auf Grund neuer Erkenntnisse globale Veränderungen vornehmen möchte. Im Extremfall wird das UI-Modell weggeworfen und von vorne begonnen.

Es werden Werkzeuge mit Planungscharakter benötigt.

Die Entwicklung einer Benutzeroberfläche ist ein iterativer und kommunikativer Prozess. Die UI-Entwicklung hat meist einen evolutionären Charakter. Das liegt daran, dass der Umfang anfangs oft noch nicht überschaubar ist und dass dem Auftraggeber viele gewünschte Eigenschaften und Bedienungsmerkmale erst auffallen, wenn er eine Vorabversion des User Interface sieht.

In allen Stadien der Modellierung können grundlegende Änderungen an Navigation, Ablauf, Dialogseitenaufbau und Verhalten notwendig werden.

Das GUI-Werkzeug und die Modellierungsmethode müssen dieser Notwendigkeit durch änderungstolerante Modelle gerecht werden. Andererseits ist es auch unabdingbar, dass der Modellierungsstand täglich möglichst stabil, in sich konsistent und verlässlich ist und nicht alles im ständigen Fluss ist. Ansonsten wird das UI-Modell zu einer unberechenbaren, amorphen Masse.

5.1 Perspektiven des UI-Entwickelns

UI-Perspektiven

Das Modellieren einer Softwareoberfläche ist ein iterativer, evolutionärer Vorgang, bei dem die Oberfläche aus unterschiedlichen Sichten heraus analysiert und definiert wird.

Eine Softwareoberfläche wird beim Schreiben wie beim Lesen der Oberflächenspezifikation aus verschiedenen Perspektiven betrachtet. Diese Perspektiven sind Sichten auf

- die Reihenfolge und Hierarchie der logischen Schritte,
- die Inhalte eines einzelnen Bildschirms,
- die Übergänge zwischen Bildschirmen und Dialogelementen,
- die Schnittstellen zu Funktionen und zu Daten der Anwendung
- und auf Zusammenhänge zwischen spezifizierten Oberflächeneigenschaften und den zugrunde liegenden Anforderungen.

UI-Perspektiven entsprechen verschiedenen Brillen, die man beim Schreiben und Lesen einer User Interface-Spezifikation aufsetzt, um die Oberfläche in ihren Zusammenhängen und Einzelheiten zu verstehen bzw. verständlich zu machen.

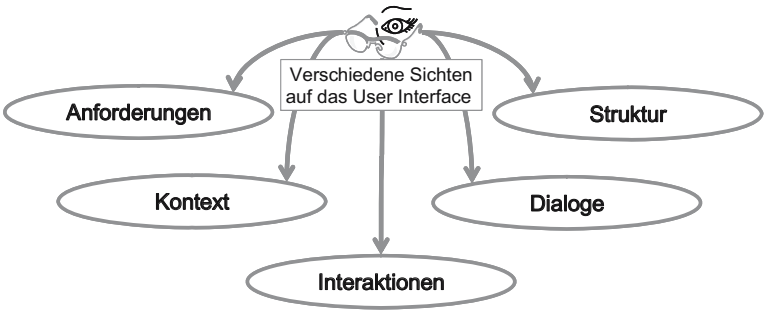


Bild 47: Verschiedene Sichten auf die Oberfläche

Die gerade benötigte Perspektive wechselt oft. Sie wird vom erreichten Fertigstellungsgrad der Spezifikation und vom inhaltlichen Interesse des Schreibers bzw. Lesers bestimmt. Die verschiedenen Perspektiven werden in der Regel durch unterschiedliche Rollen der Projektbeteiligten wahrgenommen.

Die ***Strukturperspektive*** definiert die inhaltliche Breite der Softwareoberfläche. Mit dieser Sicht beginnt üblicherweise die Arbeit an einer Spezifikation, indem eine inhaltliche Gliederung des User Interface aufgenommen wird.

Die ***Anforderungsperspektive*** legt den Zielbezug der Softwareoberfläche fest. Sie stellt den Bezug zu Use Cases, zur Featureplanung, zu funktionalen und nichtfunktionalen Anforderungen und zur Abgrenzung des Anwendungsumfangs her. Mit dieser Sicht werden zuerst die Struktur und dann die weiteren Perspektiven qualifiziert (bewertet) und validiert (bestätigt).

Die ***Dialogperspektive*** definiert die inhaltliche Tiefe der Softwareoberfläche. Es ist die Sicht auf eine abgegrenzte Dialogeinheit, z.B. eine Dialogseite oder ein Sprachmenü. Diese Sicht sollte man nach hinreichender Abgrenzung der Struktur ausformulieren.

Die ***Ressourcenperspektive*** deklariert die von anderen Sichten verwendeten Kontextbausteine der Softwareoberfläche. Diese Sicht wird implizit und explizit jederzeit dann verwendet, wenn in einer anderen Sicht eine Bezugnahme definiert wird und hierzu ein entsprechendes Bezugsziel festgelegt werden soll. In dieser Perspektive werden Schnittstellen festgelegt.

Die ***Interaktionsperspektive*** ist eine Vertiefung der Dialog- und Strukturperspektive. Sie beschreibt, an welchen Strukturstel-

len und in welchen Dialogstellen auf das Auftreten welcher Ereignisse welche Reaktionen folgen. Diese Sicht sollte man nach Festigung der Struktur und nach Abgrenzung der Dialoginhalte ausformulieren.

Im iterativen Ansatz erfolgt der Aufbau der UI-Spezifikation stufenweise durch abwechselnde Betrachtung von Anforderungen, Struktur, Dialogseiten, Ressourcen und Interaktionen in definierten Detaillierungsstufen.

In der Spezifikation einer Softwareoberfläche wirken die verschiedenen Modellierungsperspektiven

- Anforderungsperspektive,
- Strukturperspektive,
- Dialogperspektive,
- Interaktionsperspektive
- und Kontext- bzw. Ressourcenperspektive

zusammen und bilden gemeinsam den Spezifikationsstand der jeweiligen Iteration. Beim Spezifizieren werden die verschiedenen Perspektiven miteinander verwoben.

Ziel des iterativen Vorgehens ist, dass die Spezifikation mit dem Vollziehen jeder Iteration auf der für diese Iteration definierten Detaillierungsebene vollständig ist. Blockierende Gegensätze sollen erkannt und durch Designentscheidungen ausgeräumt werden.

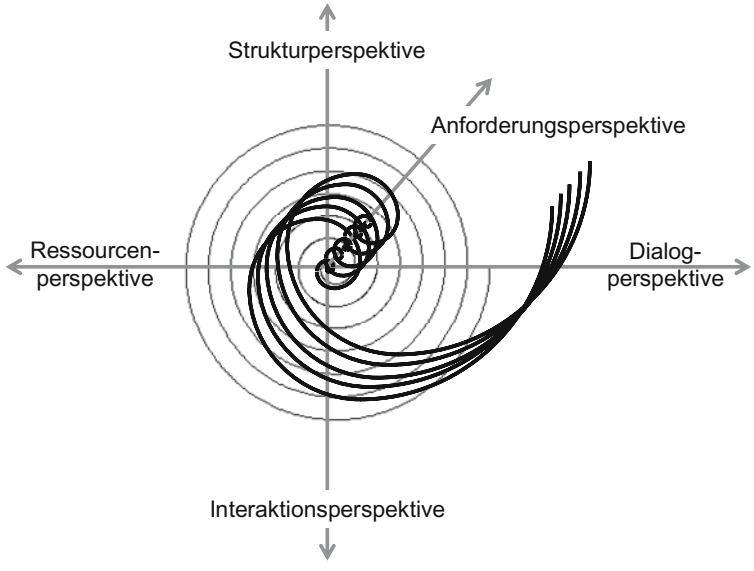


Bild 48: UI evolutionär aus wechselnden Perspektiven entwickeln

Das User Interface-Modell wird nach dem Ansatz der UI-orientierten Architektur ***gleichmäßig*** aus der jeweiligen Perspektive und aus der Wechselwirkung der Perspektiven untereinander heraus aufgebaut. Im iterativen Modellierungsprozess wird die anfangs grobe User Interface Spezifikation Iteration für Iteration über ihre gesamte Breite vertieft.

5.2 Iterationsmodell

Um die Phasen der UI Definition besser zu überblicken, bietet sich eine Einteilung des UI-Modellierungsprozesses in Fertigstellungsstufen an.

Das Vorgehensmodell RUP (Rational Unified Process) unterteilt den Software-Entwicklungsprozess in vier Phasen, in denen die verschiedenen Entwicklungsaufgaben unterschiedlich stark ausgeprägt sind. Die untenstehende Grafik verdeutlicht den Vorgehensansatz des Rational Unified Process[3].

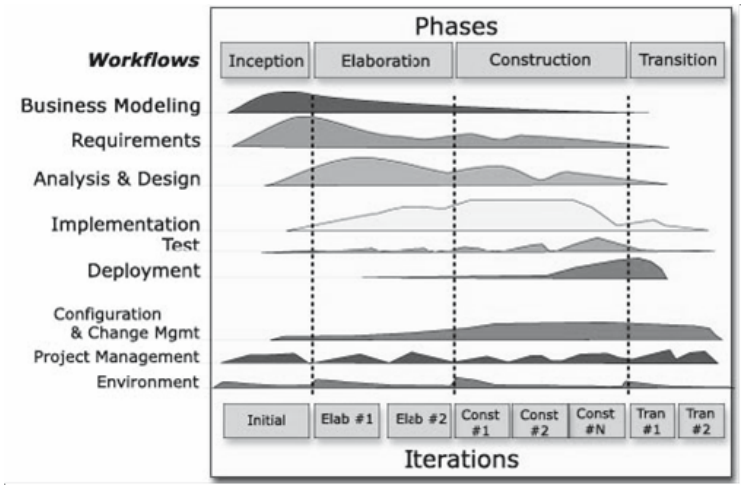


Bild 49: Rational Unified Process

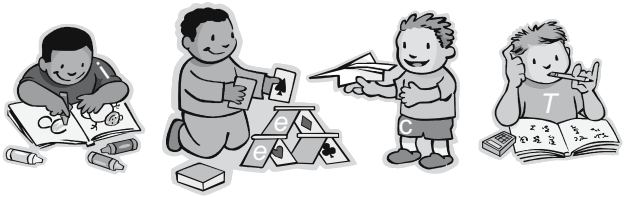
Die im RUP vorgenommene Aufteilung bietet aus meiner Sicht eine brauchbare Ausgangsbasis für das stufenweise und iterative Entwickeln einer Softwareoberfläche. Innerhalb der Phasen können die Iterationen beliebig zugeschnitten und so in das projektspezifische Vorgehensmodell (z.B. Kombination mit dem V-Modell) eingegliedert werden.

[3] Quelle: http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/jan01/WhatIstheRationalUnifiedProcessJan01.pdf

Detaillierungsstufen

In Anlehnung an die in RUP aufgestellten Entwicklungsphasen und deren Tätigkeitsschwerpunkte kann man das Modellieren eines User Interface in die Hauptiterationen ***UI-Inception***, ***UI-Elaboration***, ***UI-Construction*** und ***UI-Transition*** einteilen.

Bild 50: UI Inception, Elaboration, Construction, Transition



Den Detaillierungsstufen bzw. Hauptiterationen des UI-Entwickelns kann man wie folgt Schwerpunkttätigkeiten und Fertigstellungsgrade zuordnen.

UI-Inception ist im Kapitel Skizzieren beschrieben

- ***UI-Inception***
  - ***inhaltlich abgrenzen***
  - Umfang skizzieren
  - Gliederung skizzieren
  - Bedienungskonzept skizzieren
  - Kernanforderungen sammeln
  - Kerninhalte sammeln
- ***UI-Elaboration***
  - ***sammeln und ordnen***
  - gliedern und strukturieren
  - Detaillieren der Struktur
  - Detaillieren der Dialogsicht
- ***UI-Construction***
  - ***formen und verknüpfen***
  - Detaillieren des Verhaltens
  - Detaillieren des Bedienungskonzepts
- ***UI-Transition***
  - ***in Kontext einbetten***
  - Restriktionen ausformulieren
  - Varianz ausformulieren
  - Ausnahmen behandeln
  - Erreichen der Implementierungsreife

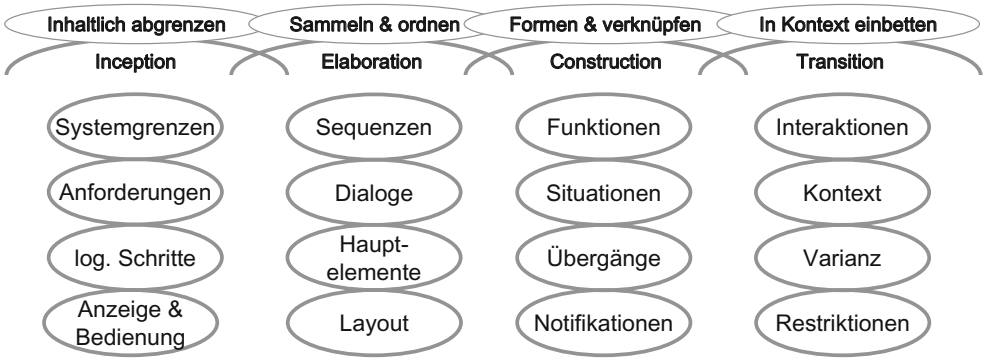


Bild 51: Iterationsmodell für die UI-Entwicklung

Die Grafik verdeutlicht, welche Themen in den einzelnen Iterationen des UI-Entwickelns geklärt werden sollten. Diese Klärung bedeutet nicht, dass die einzelnen Themen am Ende der Iteration abschließend und endgültig spezifiziert sind, sondern dass jeweils ein Fundament steht, auf dem abgesichert die weitere Detaillierung in der nächsten Iteration erfolgen kann.

In jeder Iteration werden alle Perspektiven betrachtet, jedoch mit unterschiedlichen Schwerpunkten und in verschiedener Detailtiefe.

UI-Inception: Inhaltlich abgrenzen

In der ***UI-Inception*** liegt der Schwerpunkt auf der inhaltlichen Abgrenzung des User Interface. Ziel ist, die Breite der Softwareoberfläche zuverlässig abzustecken. Mit der in der UI-Inception aufgestellten „UI-Landkarte“ kann man sicherstellen, dass man im späteren Entwicklungsverlauf nicht durch das „Entdecken neuer Länder“ überrascht wird.

UI-Elaboration: Sammeln und ordnen

In der ***UI-Elaboration*** liegt der Schwerpunkt auf dem Sammeln und dem Ordnen von Bestandteilen, aus denen sich das User Interface zusammensetzt (Ablaufstruktur, Kontrollelemente, Verhaltenselemente). Ziel ist, dass das Gefüge aus Dialogen mitsamt ihrer Anzeige- und Bedienelemente sowie die Arbeitsabäufe möglichst vollständig bekannt sind.

UI-Construction: Formen und verknüpfen

In der ***UI-Construction*** liegt der Schwerpunkt auf dem Formen und Verknüpfen der User Interface-Bestandteile. Dazu gehört das Formulieren von Übergängen zwischen Dialogeinheiten und das Benennen der Bedingungen für diese Übergänge. Ziel ist, dass das UI für den Anwender anfassbar und erlebbar wird und anhand einer Simulation bestätigt werden kann.

UI-Transition: In Kontext einbetten

In der ***UI-Transition*** liegt der Schwerpunkt auf dem Verknüpfen des User Interface mit der Funktionalität und mit dem Datenkontext. Dazu gehört das Ausdetaillieren von Verhaltensbedingungen, Nebenpfaden, Ausnahmenbehandlung, Besonderhei-

ten der Bedienung und Implementierungsdetails. Ziel ist das Erreichen der Umsetzungsreife in der Zielplattform.

In einem Modellierungsleitfaden kann jeder UI-Iteration zugeordnet werden, welche UI-Eigenschaften darin im Fokus stehen und welche Eigenschaften als bereits vorliegend und weitgehend stabil vorausgesetzt werden. Mit der Detaillierung des Leitfadens bis zur Ebene einzelner UI-Eigenschaften erhalten die Modellierer eine genaue Verfahrensanleitung.

Weitere Anregungen zum Aufstellen eines projektspezifischen Modellierungsleitfadens liefert der Vergleich des Iterationsmodells der User Interface-orientierten Architektur mit den fünf Stufen des Produktionsprozesses für Interaktive Web Software und für Hypertext Websites von Jesse James Garret. Nach Garret (siehe auch http://www.jjg.net/elements/pdf/elements.pdf) hat das Web eine duale Verwendung als Software Interface und als Hypertext-System.

Die von Khazaeli [Khazaeli 2005] in Anlehnung an Garett beschriebenen fünf Produktionsstufen sind:

- ***Strategie***
  - Ziele der Anwendung bzw. der Site
  - Anforderungen der Benutzer
- ***Scope*** (Umfang)
  - Inhaltliche Anforderungen
  - Funktionale Spezifikationen
- ***Skeleton*** (Gerüst)
  - Interaktionsdesign (im Sinne von Dialogfluss und von Navigationselementen)
  - Informationsarchitektur (im Sinne der Strukturierung, Gliederung und Vernetzung der Informationsknoten)
- ***Structure*** (Struktur)
  - Informationsdesign (Präsentation von Informationen in möglichst verständlicher Form)
  - Navigationsdesign (Elemente zum Navigieren durch die Informationsarchitektur)
  - Interface Design (Bedienungselemente zur Unterstützung der Interaktion zwischen Anwender und Anwendungsfunktionalität)
- ***Surface*** (sinngemäß Oberflächenbeschichtung)
  - Visuelles Design: Grafische und visuelle Repräsentation der Bedienungselemente, Navigationselemente und Textelemente (Look-and-Feel)

UI-Iterationen der User Interface-orientierten Architektur und Web-Produktionsstufen weisen Parallelen auf.

- Die Produktionsstufen „Strategie“ und „Scope“ sind mit der ***UI-Inception*** vergleichbar.
- „Skeleton“ ist in Teilen mit ***UI-Elaboration*** vergleichbar.
- „Structure“ ist in Teilen mit ***UI-Construction*** vergleichbar.
- „Surface“ kann als Teil von ***UI-Transition*** angesehen werden.

UI-Iterationen der User Interface-Architektur gehen (vor allem in der Iteration „UI-Transition“) inhaltlich über die Web-Produktionsstufen hinaus, z.B. bei der Kontexteinbettung (Situationen, Restriktionen, Daten- und Funktionsbezug, Varianz).

Weitere technisch orientierte Detaillierungsaufgaben bei der UI-Construction und bei der UI-Transition sind:

- ***Parametrisieren*** der Kontexte für Darstellung von Widgets und für Funktionsaufrufe.
- ***Klassifizieren*** der Dialogübergänge in Haupt- und Nebenpfade.
- ***Mapping-Angaben*** für die Kopplung des UI mit der Applikationsschicht (Daten- und Funktionsdeklarationen)
- ***Wiederverwendungsstrategien*** (Zusammenfassen von ähnlichen Eigenschaften in Musterschritten, Extrahieren von Situationen)

Iterationsabschluss

Auf Grundlage der obigen Iterationseinteilung kann die UI-Entwicklung in mehreren, klar festgelegten „Waschgängen“ durchgeführt werden. Jeder Waschgang hat das Ziel, die der Iteration zugeordneten UI-Eigenschaften über das gesamte Modell (d.h. auch aus allen UI-Perspektiven) gleichmäßig auf einen bestimmten Stand zu bringen.

Man erhält so einen stabilen Ausgangsstand für den nächsten Iterationsschritt und kann kontinuierlich den Fertigstellungsgrad, den Arbeitsfortschritt und die Qualität des UI-Modells messen.

Durch den Abschluss einer UI-Entwicklungsiteration wird der Meilenstein erreicht, an dem das UI-Modell den für diese Iteration aufgestellten Kriterien genügen soll.

Wichtige Kriterien für einen Iterationsabschluss sind:

- Definierte Umfänge sind erreicht
- Definierte Ausgangskriterien wurden erfüllt
- Definierte Eingangsbedingungen für die nächste Iteration liegen vor und können erfüllt werden

- Vorliegen von geforderten Modellteilen
- Vorliegen von Verknüpfungen zwischen Modellelementen
- Gleichmäßige Modellierungstiefe

Typische Tätigkeiten am Ende einer Iteration sind:

- Angeben aller geforderten Attribute pro Kontrollelement
- Angeben aller geforderten Attribute für jeden logischen Schritt
- Eintragen des Fertigstellungsgrads in die Modellelemente
- Absichern der Ergebnisse durch Begutachtung und Bestätigung seitens der Vertreter des Auftraggebers (einschließlich Anwendervertreter)

Wie viele UI-Iterationen in einem konkreten Projekt stattfinden und wie sie im Detail voneinander abgegrenzt werden, hängt von der Projektgröße, vom Vorgehensmodell des Herstellers, vom Vorgehensmodell des Auftraggebers und von der Abstimmung der Vorgehensmodelle aufeinander ab. Ausschlaggebend ist, dass man im Vorfeld weiß, woran der Erfolg der Iteration gemessen wird. Nur dann ist ein sachlicher Soll-Ist-Vergleich praktikabel.

5.3 Fertigstellungsgrad messen

Referenzielle Integrität

Alle in der UI-Beschreibung referenzierten Dinge, z.B. logische Schritte, zu denen in einer Interaktion verzweigt wird, müssen definiert sein.

Quantitative Messungen

Die quantitativen Messungen können durch Zählen von logischen Schritten und Kontrollelementen vorgenommen werden. Die statistische Analyse, z.B. wie viele Kontrollelemente hat ein Dialog durchschnittlich und welche Ausreißer gibt es in der UI-Beschreibung nach oben und nach unten, liefert Anhaltspunkte für Aussagen über die Gleichmäßigkeit und damit über den Fertigstellungsgrad bzw. über den Umfang des UI-Modells.

Weitere Ansätze für das quantitative Messen des Fertigstellungsgrades finden Sie im Kapitel „Testen“.

Qualitative Einschätzungen

Die qualitativen Einschätzungen erfolgen durch Bewertung von Stichproben. Dabei werden gleichermaßen die Ergebnisse der quantitativen Messungen als auch subjektive Einschätzungen der Projektbeteiligten herangezogen.

Eine Form der qualitativen Einschätzung ist die Bewertung, wie stabil (also so beschrieben, dass keine grundlegenden Veränderungen mehr erwartet werden) sind einzelne logische Schritte, die Ablaufstruktur, einzelne Dialoge und Kontext. Als Maßzahlen für solche qualitativen Einschätzungen bieten sich prozentuale Aussagen an, z.B. „Die Qualität von ‚Kundendaten erfassen' beträgt 80%".

Es empfiehlt sich, vorab festzulegen, welche Prozentangaben plausibel sind und was sie bedeuten. Zum Beispiel ist die Angabe „zu 95% fertig" in der Regel nicht weiter verwertbar, weil die restlichen fünf Prozent oft einen überraschend langen Zeitraum in Anspruch nehmen.

Das sollte man nicht verwechseln: Durch die Prozentangabe wird nicht die hineingesteckte Arbeit, sondern der Arbeitsstand des UI-Modells ausgedrückt.

Eine mögliche Prozentskala zur ***Bewertung des Fertigstellungsgrades*** könnte lauten:

Grad der Fertigstellung

- 1 - 30%: Begonnen
- 31 - 60%: Sammlung und Aufbau
- 61 - 80%: Gesamtbild liegt in Breite und Tiefe vor. Keine Priorität-1-Probleme.
- 81 - 90%: Alle UI-Merkmale können (evtl. mit kleinen Umwegen) ausprobiert werden. Keine Priorität-2-Probleme.
- 91 - 100%: Optimierung. Alle Anforderungen der Priorität 1, 2 und 3 sind umgesetzt.

Die beim Bestimmen des Fertigstellungsgrads herangezogene ***Priorität von UI-Anforderungen*** und UI-Problemen kann (ähnlich einer Fehlerklassifikation) in folgende Klassen unterteilt werden.

Klassifizieren von UI-Issues

- Priorität 1: „Show Stopper" - Das UI ermöglicht keine geschlossenen Verwendungsszenarien.
- Priorität 2: „Arbeitsbehinderung ohne Umgehungsmöglichkeit" - In den Verwendungsszenarien des UI sind Lücken, die nicht umgangen werden können. Die den Lücken nachgelagerten UI-Eigenschaften können damit nicht ausprobiert werden.
- Priorität 3: „Arbeitsbehinderung mit Umgehungsmöglichkeit" - In den Verwendungsszenarien des UI sind Lücken, welche aber (z.B. durch trickreiches Anwenden anderer UI-Features) umgangen werden können. Somit kann das betreffende UI-Feature bzw. Verwendungsszenario auf Umwegen erprobt werden. Zu dieser Kategorie zählen

auch irreführende Anleitungen und sachlich falsche Beschriftungen von Eingabefeldern.

- Priorität 4: „Schönheitsfehler“ - UI-Fehler, die das Verwendungsszenario nicht beinträchtigen, z.B. Rechtschreibfehler, Grafikfehler, Aussprachefehler und Fehler im Layout von Dialogseiten.
- Priorität 5: "Verbesserungswünsche“ - Gewünschte UI-Eigenschaften, die über die für das Projekt festgelegten Verwendungsszenarien hinausgehen.

Anhand des obigen Bewertungsschemas kann für das konkrete Projekt ein nachvollziehbares Regelwerk zur qualitativen Absicherung der User Interface Entwicklung aufgebaut werden.

5.4 Nutzen der UI-Detailmodelle

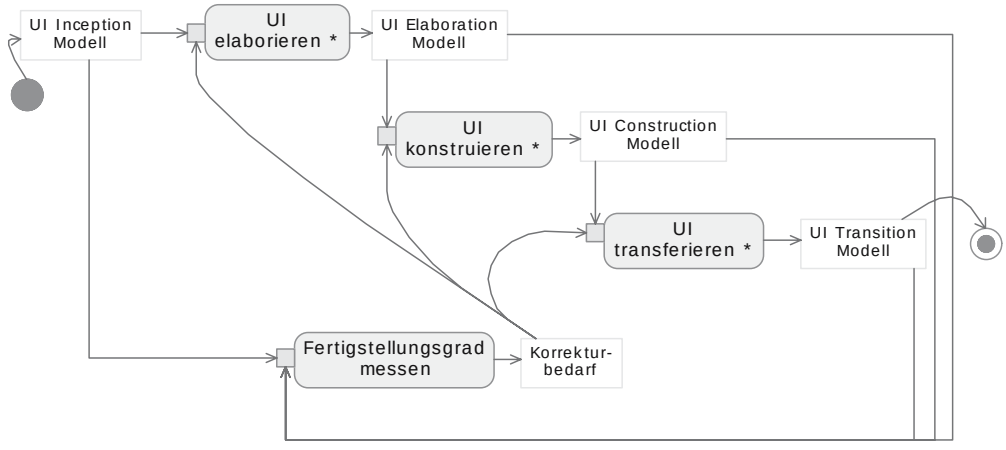
Die iterative Detaillierung der UI-Beschreibung liefert ein geschlossenes Bild der Anwendung und ermöglicht die Messung des Entwicklungsfortschritts.

Durch klar festgelegte Iterationsumfänge wird der Fertigstellungsgrad des User Interface einschätzbar und der verbleibende Aufwand bleibt in allen Entwicklungsphasen planbar. Wenn klar definiert ist, was in einer Iteration modelliert werden soll und was nicht, steht allen Beteiligten ein nachvollziehbarer Leitfaden für ein geordnetes gemeinsames Vorgehen zur Verfügung.

So kann jeder Projektbeteiligte einschätzen, wo die UI-Entwicklung steht und was der nächste Schritt ist, einschließlich der Klarheit darüber, wann die UI-Spezifikation wirklich fertig ist.

Bild 52: Leitfaden für das Detaillieren der UI-Beschreibung

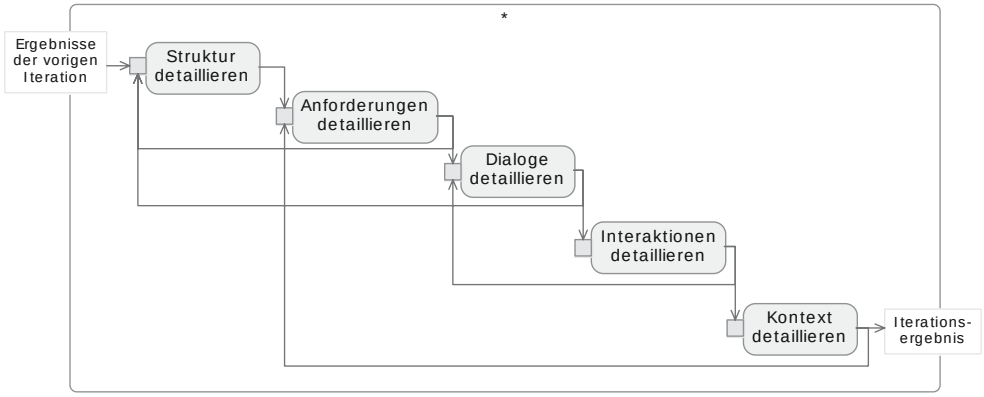
Im Kapitel Detaillieren habe ich Ihnen eine Vorgehensweise für das iterative Detaillieren einer Oberflächenbeschreibung vorgestellt. Das folgende Diagramm fasst die vorgestellten Tätigkeiten und den Arbeitsfluss beim Detaillieren eines UI-Modells zusammen.



Ziel des oben abgebildeten Leitfadens ist das Erreichen des Zustands „***Implementierungsreifes Modell***“. Die Aussage, dass ein implementierungsreifes Modell vorliegt, ist dann tragfähig, wenn zwischen Spezifizierern und Programmierern Konsens darüber besteht, dass die UI-Beschreibung exakt genug ist, um über die verbleibenden Details auf Implementierungsebene plausible Annahmen zu treffen.

Reif für die Implementierung: Exakt genug, um plausible Annahmen zu treffen

Bild 53: Ablauf einer UI-Iteration



Der ausschlaggebende Nutzen eines Vorgehensmodells zur iterativ-evolutionären Detaillierung der UI-Beschreibung ist die Orientierungshilfe, die es bietet. Das Iterationsmodell stellt eine sichere Bemessungsgrundlage für die Planung, Durchführung und Koordination der Tätigkeiten bei der Entwicklung einer Oberfläche. Für jede Modellierungsiteration wird klar festgelegt, welche Erwartungshaltung an das Modellierungsergebnis gestellt werden kann.

Die iterative Detaillierung der UI-Beschreibung bietet einen Entwicklungsweg, der integrierend und deeskalierend wirkt, indem er die Konzentration auf Sachthemen der User Interface-Entwicklung unterstützt.

Kapitel 6: Sprache

… in dem eine Sprache zur eindeutigen Beschreibung von Softwareoberflächen vorgestellt wird.

Kapitelziele

- *Wissen, wie und womit man die gesamte Oberfläche eindeutig beschreiben kann.*
- *Beispiele für typische UI-Modellierungssituationen.*
- *Lösungen für typische UI-Modellierungsprobleme.*

In den vorangegangenen Kapiteln wurden verschiedene Methoden und Werkzeuge zum Beschreiben von User Interfaces vorgestellt.

In den Kapiteln „Skizzieren“ und „Detaillieren“ haben Sie Vorgehensleitfäden kennen gelernt, die Sie beim Beschreiben von Softwareoberflächen unterstützen können.

Auf dieser Grundlage können Sie mit formalen Modellierungselementen (z.B. Step, Screen, Interaction) das User Interface einer Softwareanwendung skizzieren. Die aufgezeigten Methoden und die User Interface-Perspektiven ermöglichen Ihnen ebenfalls das Detaillieren der Skizzen zu einer weitgehend vollständigen Beschreibung des User Interface.

Damit stehen Ihnen hilfreiche Ausdruckmittel zur UI-Beschreibung und ein Leitfaden für deren Anwendung zur Verfügung. Die vorgestellten Möglichkeiten können Sie zum Modellieren von Bildschirmabfolgen, der Kontrollelemente und des Verhaltens einer Anwendung nutzen.

Sprachmodell zur UI-Beschreibung

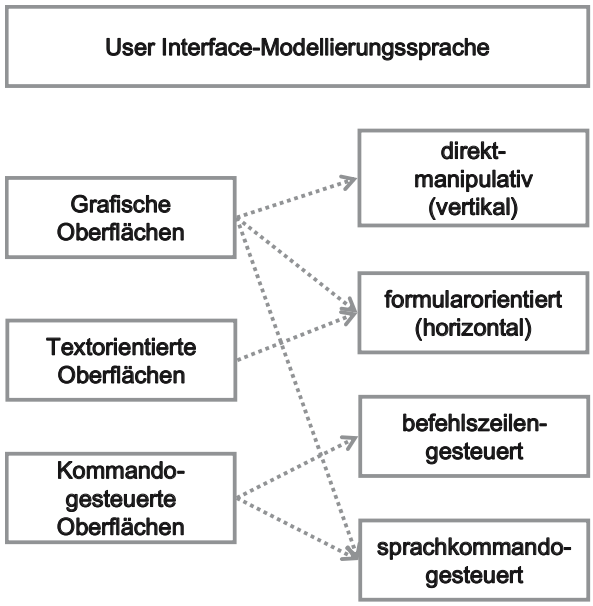
Aus praktischen Gründen habe ich bislang nur ausgewählte Teile einer Oberflächenbeschreibung ausführlich beleuchtet und auf ein vollständiges Modell für User Interfaces verzichtet.

Das Kapitel „Sprache“ schließt die noch offenen Lücken und stellt das Konzept eines ***Sprachmodells*** für User Interfaces vor, mit dem sich eine Softwareoberfläche vollständig und präzise beschreiben bzw. konstruieren lässt. Der Einsatzzweck dieses Modells ist das Spezifizieren von ***grafischen*** und ***sprachkommandogesteuerten*** Benutzeroberflächen für kommerzielle dialogintensive (vor allem horizontale[4] aber auch vertikale) Softwareanwendungen.

---

[4] Siehe Abschnitt „Horizontale und vertikale Oberflächen“ im Kapitel „Oberfläche“.

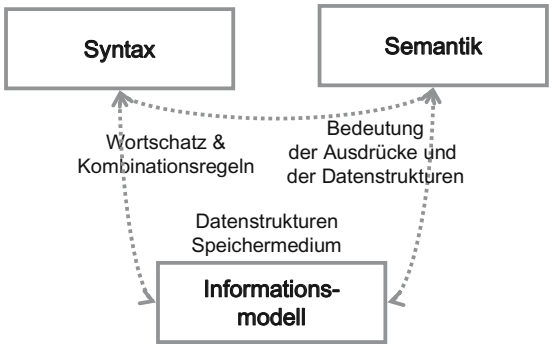
Bild 54: Einsatz einer UI-Sprache für praxisübliche Oberflächen.



Das Gesamtmodell verbindet die verschiedenen Ausdrucksmittel (Screenbeschreibung, logische Schritte, Anforderungsbeschreibung) zu einer integrierten UI-Beschreibungssprache.

Es stellt eindeutige ***Ausdrucksformen*** zum Festlegen von UI-Eigenschaften und eine ***Datenstruktur*** zur Verwaltung dieser Informationen bereit.

Bild 55: Syntax, Semantik, Informations-Modell



- Die ***Syntax*** einer Sprache legt den zur Verfügung stehenden Wortschatz und die Grammatikregeln für Wortkombinationen und für das Aufbauen von Sätzen in der Sprache fest.
- Die ***Semantik*** einer Sprache ist die Bedeutung von syntaktisch korrekten Worten, Wortkombinationen, Sätzen und Satzfolgen.

- Ein ***Informationsmodell*** ist eine Datenstruktur, in der die in der Sprache formulierten Informationen für eine weitere Verarbeitung (z.B. durch Auswertungsmechanismen und Transformatoren) abgelegt werden können.

Zielgruppe

Die Idee einer Spezifikationssprache und einer Datenstruktur für das Ordnen der UI-bezogenen Informationen adressiert vor allem die Probleme derer, die User Interfaces spezifizieren (Spezifizierer) und derer, die diese Spezifikationen interpretieren müssen (Programmierer).

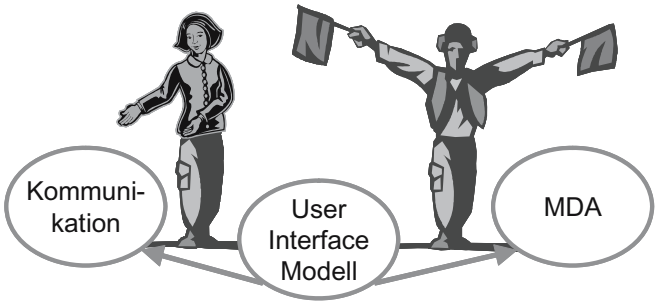


Bild 56: Von einer Beschreibungshilfe zum MDA-Werkzeug

Ziele für MDA-gerechte UI-Modelle: Eindeutigkeit, Durchgängigkeit, Vollständigkeit

Ein Sprachmodell für User Interfaces ermöglicht die Weiterentwicklung der UI-Modellierung von einem ***Beschreibungs- und Kommunikationshilfsmittel*** für UI-Anforderungen zu einer ***UI-Entwicklungswerkbank*** im Sinne der modellgetriebenen Architektur (MDA[5]).

6.1 Use Cases des UI-Modellierens

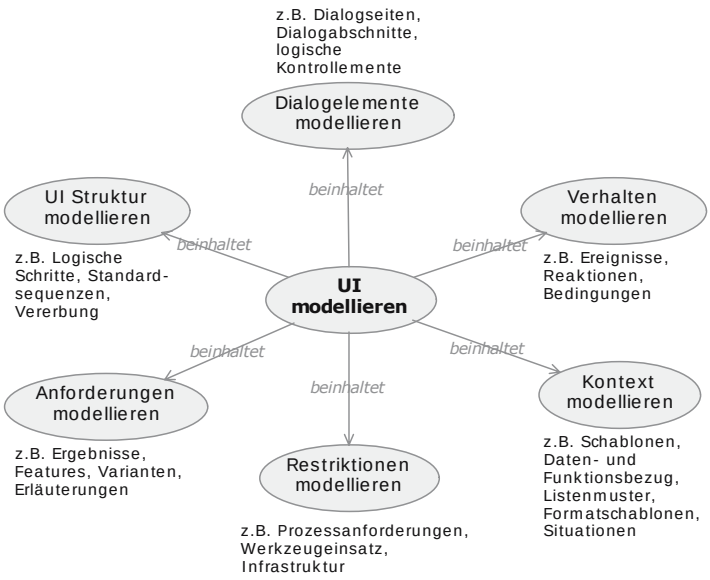
Um das Informationsgeflecht eines UI zu spezifizieren, werden Modellierungselemente benötigt, die Ausdrucksmöglichkeiten zum Beschreiben der UI-Bestandteile und ihrer gegenseitigen Beziehungen bereitstellen.

Anwendungsfälle (oder „Use Cases“) der UI-Entwicklung sind Tätigkeiten, die bei Entwurf, Design, Realisierung und Inbetriebnahme einer Benutzeroberfläche anfallen.

Die Aufteilung der Anwendungsfälle beim Entwickeln eines UI entspricht der thematischen Aufteilung einer UI-Beschreibung in UI-Perspektiven.

[5] Weitere Informationen zur MDA finden Sie z.B. auf den Seiten der OMG (http://www.omg.org).

Bild 57: Anwendungsfall-Gruppen beim UI-Modellieren



Die Grafik verdeutlicht, welche Klassen von Anwendungsfällen beim Modellieren einer Softwareoberfläche vorkommen.

Zum Großteil sind diese Tätigkeiten natürlich analog zu sonstigen Entwicklungstätigkeiten, wie z.B. „Abläufe modellieren“. Beim Detaillieren der Tätigkeiten tauchen aber User Interface-spezifische Use Cases auf, z.B. „Navigationskonzept festlegen“ oder „Dialogseite XY entwerfen“.

Rollenbezug

Verschiedene Personen (die am Projekt beteiligten Rollen) interessieren sich für unterschiedliche Teile des UI-Modells.

Der Techniker entwickelt ein Produkt von innen nach außen, der Anwendervertreter und der Spezifizierer hingegen betrachten das Produkt von außen und treffen Annahmen über seinen inneren Aufbau.

- ***Anwendervertreter*** interessieren sich für die Oberflächeninhalte und für das Verhalten der Oberfläche.
- Das Augenmerk der ***Spezifizierer*** liegt auf den Interaktionen und den Regeln, die das Verhalten der Oberfläche bestimmen.
- Im Fokus der Aufmerksamkeit der ***Programmierer*** liegt das Zusammenspiel der Interaktionen mit den Systemfunktionen und mit dem Datenhaushalt der Anwendung.

Eine vollständige Sprache zur UI-Beschreibung muss daher den unterschiedlichen Sichtweisen dieser Zielgruppen bei der Modellierung einer Oberfläche genügen.

- Anwendervertreter geben die dem Benutzer zugewandten UI-Eigenschaften vor.
- Spezifizierer modellieren die UI-Eigenschaften und UI-internen Zusammenhänge.
- Programmierer modellieren die dem System zugewandten UI-Aspekte.

Man kann zwischen Use Cases unterscheiden, die sich auf UI-Außensicht und solchen, die sich auf UI-Innensicht beziehen.

Die ***UI-Außensicht*** umfasst UI Use Cases, die sich auf Design, Usability, Ergonomie und Erleben des ***Anwenders*** richten.

Die ***UI-Innensicht*** umfasst Use Cases, die sich auf Informationsstrukturen, Beziehungen, Abhängigkeiten, Verflechtungen und Erleben des ***Entwicklers*** richten.

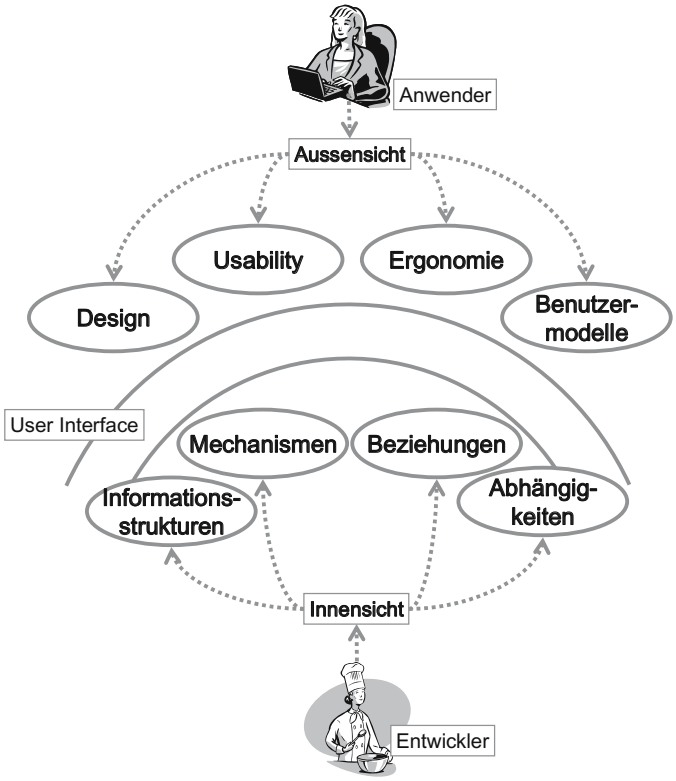


Bild 58: UI-Außensicht und UI-Innensicht

Die beiden UI-Sichten haben unterschiedliche Ausdrucksformen. Die UI-Außensicht wird mit kognitiven Modellen, durch Grafikdesign und Usability-Analysen behandelt.

Die UI-Architektur geht auf die UI-Innensicht ein.

In der UI-orientierten Architektur geht es vorrangig um das Zusammenspiel der verschiedenen Perspektiven auf die UI-

Innensicht. Die Sprache zum Beschreiben von User Interfaces stellt also Ausdrucksmittel zum Beschreiben der Innensicht (der Sicht des UI-Konstrukteurs) auf ein User Interface aus dem Blickwinkel verschiedener UI-Perspektiven.

UI-Perspektiven lösen das Problem, wie man beim Spezifizieren die Abhängigkeiten zwischen

- Anforderungen,
- Ablaufstruktur,
- der wahrnehmbarer Erscheinung von Dialogseiten,
- dem Verhalten
- dem Kontext
- und den Restriktionen

einer Softwareoberfläche in gegenseitiger Verknüpfung beschreibt. Mit den UI-Perspektiven steht ein Modell für das Ordnen und das Verknüpfen dieser Informationen zur Verfügung. Somit bieten sich UI-Perspektiven auch als eine Gliederung für eine UI-Beschreibungssprache an.

Oberfläche schrittweise entwickeln

Beim Entwickeln einer Softwareoberfläche sind zeitlich parallel zwei Fragestellungen zu klären.

- Wie erreicht man in der UI-Beschreibung schrittweise den Informationsgehalt für eine vollständige Implementierung der Oberfläche?
- Wie klärt man in der UI-Beschreibung schrittweise die fachspezifische Bedeutung der einzelnen UI-Bestandteile?

6.2 Die State Charts Notation

Eine gute Ausgangsbasis für das Aufstellen einer formalen Beschreibungssprache für Softwareoberflächen liefern auch die State Charts. Die im Kapitel „Werkzeuge“ vorgestellten UI-Modellierungsplattformen „Guide“ und „Teledrive“ enthalten jeweils einen State Chart Editor.

State Charts (Zustandstafeln) sind eine Erweiterung von Zustandsautomaten. In beiden Konzepten geht es um die Beschreibung von Maschinenzuständen. State Charts erweitern das Konzept der ursprünglichen Mealy- und Moore-Zustandsautomaten um hierarchische Vererbung und um die so genannte Orthogonalität (Gleichzeitige Existenz von mehreren Zustandsabfolgen).

State Charts werden oft als eine Standardnotation zur UI-Beschreibung angeführt [Ceri 2003], [Shneiderm. 2002]. Der ursprünglich von David Harel als Erweiterung von Zustandsauto-

maten erschaffene Formalismus der State Charts [Harel 1987] ist Teil der UML und wird dort Zustandsdiagramm genannt.

Bemerkenswert ist, dass die State Charts nicht in Konkurrenz mit anderen Formalismen zum Beschreiben von Abläufen in einem User Interface stehen. Sie wurden seit ihrer Erfindung nicht weiter für die Oberflächenentwicklung spezialisiert und entwickelt, sondern sind seit über 20 Jahren in ihrer Ursprungsform stabil geblieben.

Diese Stagnation steht ganz im Gegensatz zur Daten- und Funktionsmodellierung, wo der Wettbewerb der Modellierungsmethoden die Konvergenz und die Erweiterung der Formen antreibt.

Die State Chart Notation bietet eine Basis für die Beschreibung von Abläufen und Interaktionen in einem User Interface. Das Informationsgeflecht einer UI-Beschreibung geht jedoch über die Ausdrucksmöglichkeiten von State Charts hinaus (vgl. Abschnitt „Leitfragen der UI-Beschreibung“ im Kapitel „Oberfläche“).

Es folgt eine kurze Analyse der Ausdrucksmittel der State Chart Notation. Ein Schwerpunkt dieser Analyse ist, wie gut sich die aus den Kapiteln „Oberfläche“, „Skizzieren“ und „Detaillieren“ bekannten Perspektiven einer UI-Beschreibung durch State Charts abbilden lassen.

Elemente der State Charts Notation

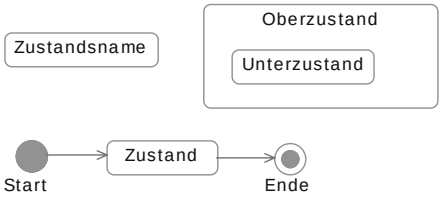
Die zentralen Elemente der State Chart Notation sind der ***State*** (Zustand) und die ***Transition*** (Übergang).

State Charts haben folgende Notationselemente:

- Zustandstafel (Hülle einer State Chart)
- Zustände
- Start- und Endzustände
- Pseudozustände (z.B. Kreuzung und Vereinigung)
- Transitionen (Übergänge mit Auslösern, Bedingungen und Reaktionen)
- Zustandsaktivitäten

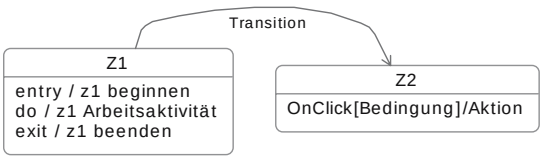
Der Zustandsname wird in einem Rechteck mit abgerundeten Ecken angeschrieben. Zustände können ineinander geschachtelt werden. Der Start- und der Endzustand werden durch besondere Symbole dargestellt.

Bild 59: Zustände in einer State Chart



Zustände können festgelegte Eintritts- und Austrittaktivitäten, Arbeitsaktivitäten sowie weitere, durch dedizierte Auslöser festgelegte Aktivitäten haben.

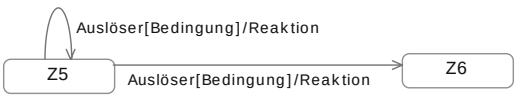
Bild 60: State Chart: Zustände mit Aktivitäten und Transitionen



Die Übergänge können zwischen zwei Zuständen erfolgen oder zustandsintern sein. Ein Übergang kann einen Auslöser, eine Bedingung und eine Aktion haben. Der Auslöser ist ein Ereignis, mit dem das Abarbeiten des Übergangs beginnt. Die Bedingung legt (z.B. in Form eines logischen Ausdrucks) fest, welche weiteren Begebenheiten gelten müssen, damit der Übergang auch tatsächlich ausgeführt wird. Die Reaktion definiert Aktivitäten (z.B. das Ausführen von Funktionen), die im Vorfeld des Übergangs zum Zielzustand ausgeführt werden.

Der Übergang erfolgt ohne zeitliche Verzögerung, d.h. ein Übergang nimmt per Definition keine Zeit in Anspruch.

Bild 61: Auslöser, Bedingung und Reaktion in einer Transition



Trigger können Transitionen bzw. Aktivitäten auslösen. Es gibt folgende Trigger:

- Signal Trigger
- Call Trigger
- Time Trigger
- Change Trigger

Mehrere Zustände, d.h. ihre Transitionen und Aktivitäten in abgetrennten Tafelteilen können gleichzeitig ausgeführt werden. Orthogonale Zustände beziehen sich in der Regel auf unterschiedliche Objekte.

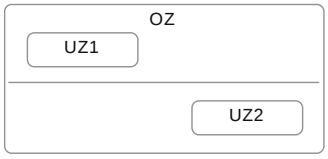


Bild 62: Orthogonale Zustände

Mit der flachen und der tiefen Historie können Unterzustände gemerkt werden. Dadurch kann z.B. festgelegt werden, dass beim erneuten Betreten eines übergeordneten Zustands der zuvor aktive Unterzustand eingestellt wird.

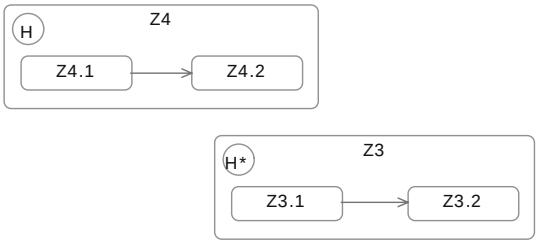


Bild 63: Flache und tiefe Historie

Die untenstehende Grafik zeigt ein Beispiel für eine State Chart.

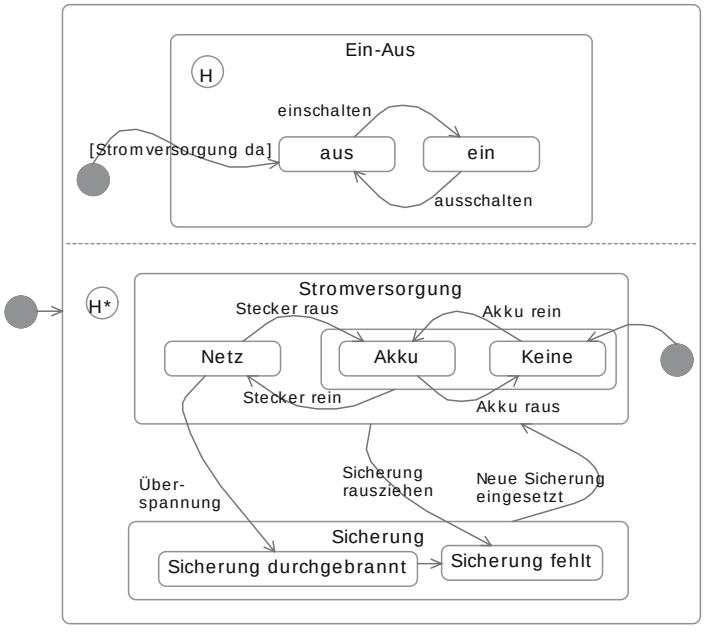


Bild 64: State Chart für einen Schalter

Das Diagramm verdeutlicht die Zustandsübergänge beim Ein- bzw. Ausschalten eines elektrischen Geräts (z.B. eines Laptops).

Das Gerät hat eine Überspannungssicherung und kann per Netzstrom oder per Batterie betrieben werden.

Schwächen von State Charts und Lösungsansätze

Im Abschnitt „Typische Phänomene des Spezifizierens“ wurde vorgestellt, welche Standardanforderungen an die Spezifikationsmethode beim Beschreiben einer Benutzeroberfläche auftreten.

Diese Standardanforderungen betreffen folgende Phänomene:

- Ablaufbeschreibung
- Vererben und wieder verwenden
- Transitionslogik
- Kontextabhängige Präsentationslogik
- Lokalisieren

Eine Methode zum Spezifizieren von Benutzeroberflächen sollte daher die aus diesen Phänomenen resultierenden Anforderungen berücksichtigen und Handhabungsmöglichkeiten dafür bieten. Nach meiner Erfahrung mit dem Spezifizieren von Oberflächen für verschiedenste kommerzielle Anwendungen lassen sich UI-Spezifikationen nur zum Teil bzw. nicht effizient mit State Charts handhaben.

Wesentliche Schwächen der State Charts Notation beim Einsatz für das Beschreiben von User Interfaces können sein:

In Bezug auf Ablaufbeschreibung und Transitionslogik:

- Es gibt keine ***Standard Lese- und Ausführungsreihenfolge*** einer State Chart. Die Reihenfolge der Zustände einer State Chart wird durch die Transitionen bestimmt.

Kanonische Sequenzen

  - Lösungsmöglichkeit: Einführen von Standardsequenzen, die der Definitions- und Lesereihenfolge folgen.
  - Erweitern der Zustände zu Ausführungsschritten.

- Beim Lesen einer State Chart kann man nicht auf Anhieb sehen, welche Transitionen die ***Hauptpfade*** sind und welche Transitionen Ausnahme- bzw. ***Nebenpfade*** wiedergeben.

Sequenz- und Service-Schritte

  - Lösungsmöglichkeit: Unterscheiden zwischen Sequenzschritten und Serviceschritten.
  - Unterscheiden zwischen Transitionen auf dem Hauptpfad und Neben- und Ausnahmepfaden.

In Bezug auf Präsentationslogik und Lokalisierung:

- Es können nur Abläufe und Interaktionen beschrieben werden. Es fehlt aber die Unterstützung des Definierens und des Bezugnehmens auf andere UI Teile, z.B. ***Kontrollelemente***, Dialogseiten, Funktions- und Datenkontext.
  - Lösungsmöglichkeit: UI-Perspektiven, die den Use Case-Modulen des UI-Modellierens folgen.
  - Querverweise zwischen Bestandteilen der Strukturbeschreibung und Bestandteilen der Verhaltensbeschreibung sowie den Anforderungen.

Perspektiven und Querverweise

In Bezug auf Vererben und wieder verwenden:

- Eine State Chart wird durch die vielen Transitionen und Interaktionen einer modernen grafischen Oberfläche schnell komplex und ***unübersichtlich***.
- Es fehlt ein universell verwendbarer Wiederverwendungsmechanismus für Eigenschaften von States und für Transitionen.
  - Lösungsmöglichkeit: Auslagerung von Situationen, auf die bei Transitionen Bezug genommen wird, in eine eigene Modellierungssicht.
  - Bezugnahme auf bereits definierte Schritte mit einem based-on-Konstrukt.

Zentral definierte Ressourcen

6.3 Konzepte für eine UI-Beschreibungssprache

Es wäre praktisch, die schon bestens bekannten UI-Perspektiven mit der State Charts Notation modellieren zu können.

Statecharts bieten jedoch keine Unterstützung zum Aufteilen der UI-Beschreibung in Perspektiven und beim gegenseitigen Verknüpfen von Darstellung, Verhalten und Kontext.

Unter diesem Vorzeichen folgt die Analyse, welche Erweiterungen an State Charts vorgenommen werden sollten, damit UI-Perspektiven abgebildet werden können.

Die Analyse liefert die Konzeptgrundlage für eine UI-Beschreibungssprache, die

- die Gliederung der Beschreibung in UI-Perspektiven ermöglicht,
- eine Handhabe für typische Phänomene des UI-Spezifizierens bietet,

Ziel: UI-Perspektiven abbilden und typische Phänomene, die beim Spezifizieren auftreten (z.B. Varianz) handhaben.

- und an die Ausdrucksmöglichkeiten von Statecharts angelehnt ist.

Hierzu erweitere ich die Semantik der State Charts da, wo sie die oben aufgezeigten Schwächen aufweisen.

UI-Perspektiven abbilden

Statecharts bieten keine Notation für das Aufteilen einer User Interface-Spezifikation in verschiedene Sichten (Anforderungen, Ablaufstruktur, Dialoge, Interaktionen, Kontext). Üblicherweise werden diese Sichten in einer State Chart nicht getrennt. Das führt schnell zu komplexen und unübersichtlichen Diagrammen.

Zur Abbildung der UI-Perspektiven in State Charts schlage ich die Einführung von je einer typisierten State Chart pro Perspektive sowie spezielle Relationen zur Bezugnahme auf die Elemente dieser spezialisierten State Charts.

Verbund aus spezialisierten State Charts

Ein UI-Modell besteht diesem Vorschlag nach aus

- einer „Anforderungen Chart“,
- einer „Ablaufstruktur Chart“,
- einer „Dialog Chart“ pro Screen/Dialog,
- einer „Interaktion Chart“ pro Screen/Dialog,
- einer „Interaktion Chart“ für die Ablaufstruktur“,

sowie aus Querverweisen zwischen den Elementen dieser nebeneinander liegenden Charts.

Der konkrete Lösungsvorschlag ist hier das Aufteilen einer UI-Spezifikation in die Teile

- „1x ***Struktur***“ (Baum mit Ablaufschritten),
- „n x ***Screen***“ (ein GUI für einen Bildschirm oder ein Voice UI für einen Sprachdialog) und
- „1x ***Kontext Ressourcen***“ (Informationen über Umweltdinge, auf die Bezug genommen wird).

Die Interaktionen und Anforderungen können in diese Bestandteile eingestreut sein oder orthogonal dazu in eigenen Teilen

- „***Verzeichnis der Interaktionen***“ und
- „***Verzeichnis der Anforderungen***“

liegen.

Die Anforderungsperspektive wird in der UI-Spezifikation durch Anforderungsobjekte abgebildet.

Die Struktur-, Dialog- und Interaktionsperspektiven werden durch Oberflächenobjekte abgebildet.

Die Kontextperspektive wird durch Kontext- und Ressourcenobjekte abgebildet.

Vererben und wieder verwenden

Generalisierte und spezialisierte UI-Eigenschaften

In einer Benutzeroberfläche sind Verhaltensweisen bzw. Bildschirmaufbau oft an mehreren Stellen ähnlich. Zum Beispiel soll vielleicht in mehreren Bildschirmen das gleiche oder ähnliche Menü erscheinen, eine ähnliche Statusleiste eingeblendet werden oder beim Drücken einer Funktionstaste die gleiche - eventuell zentral konfigurierbare - Aktion ausgeführt werden.

Auch kann es nötig sein, auf eintretende Ereignisse in Abhängigkeit von der Stelle, an der sich der Anwender in der Softwareoberfläche befindet, ähnlich, aber differenziert zu reagieren.

Bei anderen Ereignissen, zum Beispiel bei Ausfall von angeschlossenen Geräten eines Infotainmentsystems im Auto, bei unterschiedlich wichtigen Verkehrsmeldungen oder bei eingehenden Anrufen, ist es eventuell notwendig, an jeder Stelle der Oberfläche gleich zu reagieren, aber die Priorität der gerade durchgeführten Operation zu berücksichtigen.

Beim Anzeigen einer wichtigen Staumeldung im Navigationssystem soll zum Beispiel, wenn das vom Fahrer so eingestellt wurde, das Einblenden eingehender Anrufe unterdrückt werden.

Ein anderes Beispiel wäre das Verzögern des Einblendens eingehender Nachrichten, während der Anwender in seinem PDA (Personal Digital Assistent) eine TAN (Transaktionsaktivierungsnummer) für eine Überweisung eintippt.

Die Beispiele zeigen, dass es notwendig ist, die an einer Stelle der UI-Beschreibung definierten Eigenschaften an verschiedenen Stellen und in verschiedenen Kombinationen wieder verwenden zu können. Diese Anforderung gilt sowohl für zentrale Definitionen als auch für die im Kontext einer bestimmten UI-Stelle festgelegten UI-Eigenschaften.

State Charts ermöglichen lediglich eine hierarchische Vererbung von Transitionen. Das wird durch Schachtelung von States ineinander ausgedrückt. Ein selektives Erben aus States, die außerhalb der Schachtelung liegen, wird in State Charts nicht unterstützt. Auch das Erweitern oder Überschreiben von Eigenschaften aus beerbten States unterstützen State Charts nicht.

Hierarchische Vererbung und orthogonale Vererbung

Um das nichthierarchische „Erben von Vorbildern“ zu ermöglichen, müssen State Charts erweitert werden. Neben der "hierarchischen Vererbung" von Eigenschaften wird damit eine zweite Achse, die "orthogonale Vererbung" mittels der ***„basedOn“-Beziehung*** eingeführt.

Orthogonale Vererbung löst das Problem der Wiederverwendung von unterschiedlich kombinierten Grundeigenschaften. Diese verschmelzende Vererbung löst das Problem, dass die geerbten Eigenschaften ggf. ergänzt oder überschrieben werden müssen.

Hierzu wird in State Charts die Beziehungseigenschaft

- ***„basedOn <state-id>“***

eingeführt.

Diese Beziehung bedeutet, dass die mit „basedOn“ geerbten Eigenschaften vom erbenden State überschrieben und/oder um weitere Eigenschaften ergänzt werden können.

Für die Kombination von hierarchischer und orthogonaler Vererbung gilt:

- Event-Bekanntmachung in den Steps und in den Widgets (Suche nach Event-Handler) erfolgt erst orthogonal und danach hierarchisch auf der Elternachse.
- Beim Erben der Containments erfolgt die Verschmelzung nur aus orthogonal geerbten Inhalten. Hierarchisch wird kein Containment vererbt.

Funktions- und Datenkontext verwenden

Umweltschnittstelle mit Application Values und Function Requests

Eine Benutzeroberfläche vermittelt zwischen Anwender (Mensch) und der funktionalen Anwendung (Maschine). Die Oberfläche muss daher mit der Anwendung kommunizieren, um das Ausführen von Funktionen anzustoßen und um Informationen aus dem Datenhaushalt der Anwendung zu holen.

Die Oberfläche gibt an den Anwender Informationen aus, die sie aus den Daten der Anwendung holt. Sie stellt auch Bedienelemente zur Verfügung, die auf Anwenderaktionen reagieren. Diese Reaktionen können Aufrufe von Anwendungsfunktionen beinhalten.

Zum Beispiel:

- Auf dem Bildschirm wird von der Oberfläche eine alphabetische Liste von Orten ausgegeben. Die Liste ist ein Auswahlfeld, aus dem ein Ort ausgewählt werden kann.
- Neben der Liste gibt es auf dem Bildschirm ein Eingabefeld. Werden in diesem Eingabefeld Buchstaben eingegeben, dann wird die Liste auf die Einträge eingeschränkt, in denen die eingegebene Buchstabenfolge vorkommt.
- Dazu wird die Liste (unter Übergabe der eingegebenen Buchstabenfolge als Suchschlüssel) von der Anwendung neu geholt.

- Nach Auswahl eines Orts wird entsprechend voreingestellten Kategorien die Liste der Sehenswürdigkeiten zu diesem Ort angezeigt. Dazu wird Bereitstellung der Liste der Sehenswürdigkeiten abermals per Funktionsaufruf in der Anwendung angefordert.
- Anschließend werden die von der Funktion an der UI-Schnittstelle bereitgestellten Daten abgerufen und auf dem Bildschirm ausgegeben.

Breite Schnittstelle zwischen UI und der funktionalen Anwendung

Das Beispiel zeigt, dass die Oberfläche eine breite Schnittstelle zum Funktions- und Datenkontext der funktionalen Anwendung hat.

State Charts beschreiben Zustandsübergänge innerhalb eines geschlossenen Systems. Sie unterstützen jedoch keine Wechselwirkung zwischen zwei Systemen, von denen z.B. eines das andere verwendet.

Für das Spezifizieren von Benutzeroberflächen müssen State Charts um eine Kontext- und Ressourcen-Sicht zur Deklaration der von der Oberfläche verwendeten Funktionen und Daten erweitert werden.

Hierzu können in State Charts die auf den UI-Kontext bezogenen Konstrukte

- ***applicationValue***,
- ***functionalRequest*** und
- ***onSignal***

eingeführt werden. Sie ermöglichen das Abfragen und Deklarieren von Daten aus dem Datenhaushalt der Anwendung, das Auslösen und Deklarieren von Anwendungsfunktionen und das Reagieren auf Signale aus der Funktionsschicht.

Application Values und Functional Requests lösen das Problem der Verbindung der Oberfläche mit dem funktionalen Teil der Anwendung. Wenn die Oberfläche Daten der Anwendung abrufen und Funktionen der Anwendung auslösen kann sowie auf einfache Signale der Anwendung reagiert, ist eine ausreichende Schnittstelle zur Kommunikation zwischen Oberfläche und Funktionalität gegeben. Die Oberfläche kann damit im Kontext der Funktionen und Daten der funktionalen Schicht agieren.

Kontextabhängige Präsentationslogik

Von Bedingungen zu Situationen

Eine Benutzeroberfläche hat zur Laufzeit oft den Bedarf, sitzungsspezifische Informationen, wie z.B. dass ein Knopf schon einmal gedrückt wurde oder dass ein Passwort schon zweimal falsch eingegeben wurde, auszuwerten. Ohne Kontextbezug kann die Anwendungsoberfläche keine Abläufe abbilden.

State Charts bieten so genannte Guards („Bedingungsbewacher“) an, um das Durchführen von Transitionen an Bedingungen zu knüpfen. Falls mehrere Bedingungen ausgewertet werden müssen, können die Guards schnell unübersichtlich werden. Zudem kann man nicht auf zentral definierte Guards zurückgreifen, was bei Änderungen von globalen Bedingungen das Überprüfen und Anpassen aller Guards der betroffenen Transitionen nach sich zieht.

Die Guards einer UI-Spezifikation beziehen sich auf den Funktions- und Datenkontext der Anwendung (Außenschnittstelle), nehmen aber auch oft Bezug auf den eigenen Zustand des UI. Damit hat das User Interface auch eine Selbstschnittstelle zu einem durch sich selbst aufgebauten Kontext.

Zum Beispiel könnte es notwendig sein zu wissen, welche Dialogseite zuletzt aktiv war und welches Eingabeelement gerade den Fokus hatte. Oder es wird die Information benötigt, ob ein Bildschirm bereits betreten wurde, betretbar ist, abgeschlossen wurde.

Die nötigen Informationen können nur teilweise aus der funktionalen Anwendung bezogen werden; teilweise sind sie sitzungsspezifisch und beziehen sich auf den Zustand einzelner Bedienungselemente und logischer Schritte.

Wiederverwendbare Situationen

Zur Abbildung dieser komplexen Transitionsbedingungen schlage ich die Erweiterung des Guard-Konzepts von State Charts zur Selbstschnittstelle des User Interfaces mit wieder verwendbaren ***Situationen*** vor.

Der konkrete Lösungsvorschlag beinhaltet die Einführung der folgenden semantischen Erweiterungen:

- Formulieren von Bedingungsausdrücken als zentrale Ressourcen (z.B. als Pseudo-Zustände).
- Formulieren von Interaktionen als Tupel aus Auslöser, Situationsbedingung und Reaktionen, wobei sowohl die ganze Interaktion bedingt sein kann, als auch jede darin vorkommende Reaktion.
- Möglichkeit, den Event weiterzupropagieren, oder zu konsumieren (bei Reaktion wird das Signal im Normalfall nicht weiter propagiert).
- OnEnter und OnLeave funktionieren wie normale Event-Propagation in einem Step.
- Möglichkeit, einen Step-Exit mit „refuse exit“ abzubrechen.
- Standortabhängiges Interpretieren von Sprungzielen für Step-Übergänge.

6.4 Datenmodell für UI-Spezifikationen

Der Bedarf und Nutzen eines gemeinsamen Datenmodells für die bei der UI-Entwicklung entstehenden Informationen ist, dass man auf dieser Ebene Informationen austauschen kann.

Mit dem untenstehend aufgestellten Informationsmodell schlage ich eine Informationsstruktur vor, welche dazu geeignet ist, den Informationsgehalt einer Oberflächenspezifikation aufzunehmen.

Die nachfolgende Abbildung zeigt die grundlegenden Informationseinheiten einer Benutzeroberfläche und deren Beziehungsgeflecht.

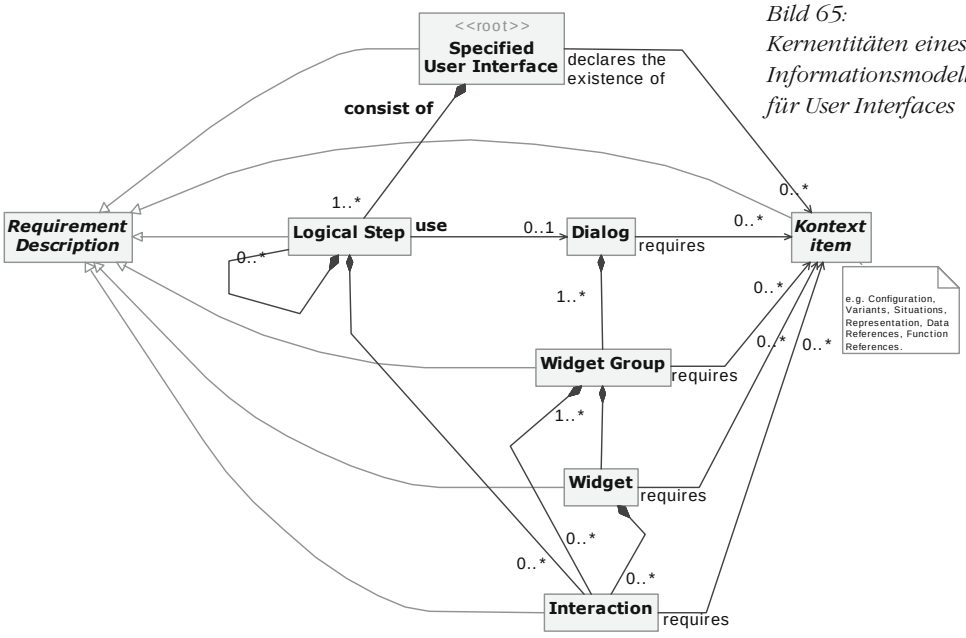


Bild 65: Kernentitäten eines Informationsmodells für User Interfaces

Mit der Aufstellung der Kernentitäten klärt das Informationsmodell für Oberflächenspezifikationen zwei zentrale Fragen:

- Welches Informationsgeflecht kann die in einer Oberfläche enthaltenen Informationen wiedergeben?
- Welche Informationen müssen in einer Sprache für Oberflächenspezifikationen formuliert werden können?

Die durch das User Interface-Informationsmodell aufgestellte Datenstruktur erfüllt mehrere Funktionen:

- Datensenke für die Aufnahme der Informationen aus der Spezifikation.
- Notationsneutrale Struktur für Objekte, Attribute und Assoziationen einer spezifizierten Benutzeroberfläche.

- Datenquelle für die Umformung der spezifizierten Informationen in verschiedene Verwendungen, z.B. in Code und Laufzeitstrukturen.

Damit bildet das Informationsmodell die **Projektionsachse** zwischen einer UI-Spezifikationssprache und den Laufzeitdatenstrukturen zum Prüfen, Auswerten und Simulieren der Anwendungsoberfläche mit Interpretationsprogrammen.

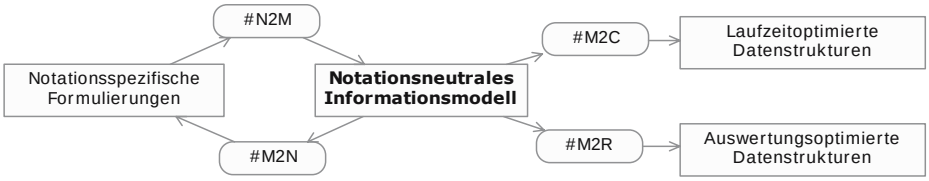


Bild 66: Das Informationsmodell kapselt Notation und Laufzeitformat

Die obige Abbildung zeigt, wie das Informationsmodell die Sprache der Spezifikation von den Datenstrukturen, auf denen die rechnergestützte Verarbeitung der Spezifikation erfolgt, kapselt. Der praktische Nutzen dieser Kapselung ist, dass die Sprache zur Spezifikationsformulierung ("Hochsprache") verändert werden kann, ohne dass diese Veränderungen auf die auf dem Informationsmodell ("Maschinennahe Sprache") aufsetzenden Auswertungswerkzeuge durchschlagen.

Die Auswertungswerkzeuge (bezeichnet durch #M2C und #M2R) werden durch die Komplexitätsreduktion bei Umwandlung von menschennahen Hochspracheformulierungen zu maschinennahen Low-Level-Strukturen (bezeichnet durch #N2M und #M2N) einfacher.

Die in der Spezifikation enthaltene Information ist primär in "Logische Schritte" bzw. „Ablaufschritte“ unterteilt. Logische Schritte legen eine hierarchische Ablaufstruktur fest, wobei jeder logische Schritt in weitere logische Schritte unterteilt sein kann.

So werden die Tätigkeiten des Anwenders in eine Reihenfolge gebracht und in kleinere Teilaufgaben gegliedert.

Logical Steps sind die **Struktur-Perspektive** der Spezifikation.

Jeder logische Schritt kann [muss aber nicht] als Kommunikationsmedium zum Anwender einen Dialog verwenden. Ein Dialog besteht aus Widget-Gruppen und aus einzelnen Widgets, die Bedienungselemente (z.B. Grafische Ein- und Ausgabe-Controls, Hardbuttons, Voicebuttons) repräsentieren.

Dialogs, Widget Groups und Widgets sind die **Dialog-Perspektive** der Spezifikation.

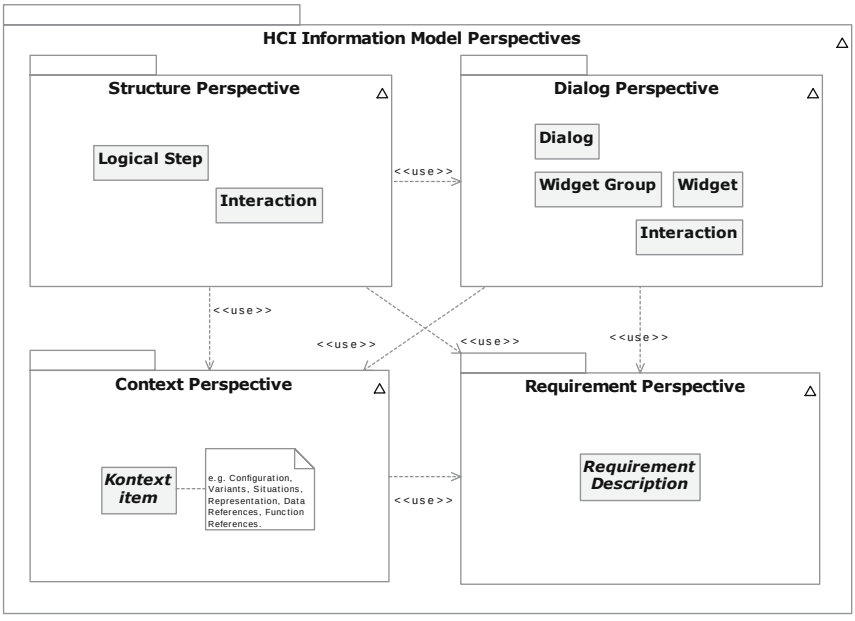
Die Oberfläche interagiert mit dem Anwender im Kontext der Funktionen und Daten der zugrunde liegenden Anwendung. Das Verhalten und die Anzeigen der Oberfläche hängen von den verfügbaren Funktionen und von Daten der Anwendung und von den Aktivitäten des Anwenders in der Oberfläche ab.

Die Kontextelemente, auf die die Oberfläche Bezug nimmt, sind die **Kontext-Perspektive** der Spezifikation.

Die Anforderungen an die Anwendung und ihre Bedienung sind Grundlage der Struktur, der Dialoge und der Kontextbezüge. Die Verknüpfung der Oberflächeneigenschaften mit den zugrunde liegenden Anforderungen erzeugt ein Geflecht, das die gegenseitige Abhängigkeit abbildet.

Die Anforderungen und ihre Verknüpfungen untereinander sowie die Verknüpfungen mit den Oberflächenbestandteilen sind die **Anforderungsperspektive** der Spezifikation.

Bild 67: Inhalte und Verflechtungen der UI-Perspektiven



Wie die obige Abbildung zeigt, sind die Perspektiven einer User Interface-Spezifikation miteinander verflochten und voneinander abhängig.

- Die Struktur der Logischen Schritte folgt den Anforderungen und benutzt Dialoge und den Kontext.

- Die Dialoge realisieren die Kommunikation der Logischen Schritte. Sie folgen den Anforderungen und benutzen den Kontext.
- Die Kontextelemente werden von Logischen Schritten und Dialogen herangezogen. Sie folgen den Anforderungen.
- Die Anforderungen bilden die Grundlage für Logische Schritte, Dialoge und Kontextelemente.

6.5 UI-Beschreibungssprache Lucia

Die in der User Interface-orientierten Softwarearchitektur vorgeschlagenen Modellierungshilfsmittel für Anwendungsoberflächen bestehen aus einer ***Informationsstruktur*** und aus einer ***Grammatik***.

Die Informations- bzw. Datenstruktur ist so beschaffen, dass sie die über eine Oberfläche gesammelten Informationen als Beziehungsgeflecht aufnehmen kann. Die Grammatik legt eine Sprache fest, mit der die Datenstruktur auf effiziente Weise befüllt werden kann.

UCIA Datenmodell
Lucia Sprache

Im Folgenden bezeichne ich diese Datenstruktur als User Centered Information Architecture bzw. als UI-Datenmodell und die Sprache als Language for User Centered Information Architecture. Die Akronyme hierfür sind UCIA und Lucia.

Der Nutzen der bisher vorgestellten Möglichkeiten zur UI-Modellierung liegt in der Bereitstellung einer Entwurfsmethodik für Anwendervertreter und Spezifizierer. Ein Ausbau der Entwurfsmethodik zu einer Modellierungssprache weitet die Gruppe der Nutznießer eines UI-Informationsmodells auf Programmierer und MDA-Entwickler aus.

Anwendervertreter, Spezifizierer und Programmierer haben verschiedene Aspekte der Software im Fokus. Die Lucia-Sprache verbindet und überbrückt die unterschiedlichen Sichten der am Projekt beteiligten Personen. Sie ermöglicht die koordinierte Zusammenarbeit der Interessengruppen, so dass sich die unterschiedlichen Sichten zu einem Gesamtbild ergänzen.

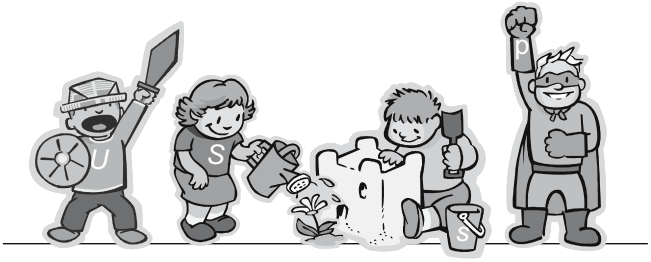


Bild 68: Spezialisten interessieren sich für unterschiedliche Softwareteile

Nutzung und Technik können so im Zusammenhang diskutiert und Lösungen auf einer gemeinsamen Formulierungsplattform erreicht werden. Die Modellierungssprache legt Zielkonflikte offen und dient zugleich als Katalysator für die Formulierung von Lösungen, die sowohl der Anwender- als auch der Techniksicht genügen.

Ziel des Modellierens in Lucia ist ein vollständiges und in sich stimmiges UI-Modell, das den Bedarf aller beteiligten Interessengruppen abdeckt und missverständnisfrei die Anwendungsoberfläche festlegt.

Das Ergebnis ist ein Modell mit eindeutiger Bedeutung der Inhalte und Relationen und kann als Datenstruktur (z.B. in XML) für die parserunterstützte Weiterverarbeitung bereitgestellt werden.

Eigenschaften der Lucia-Sprache

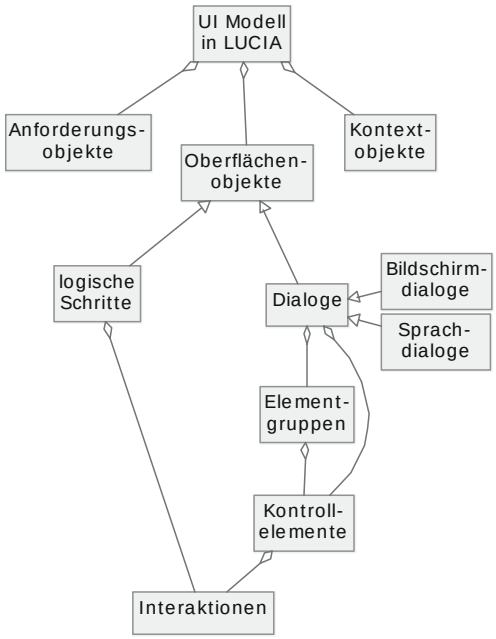
Die zur Formulierung von unseren User Interface-Beispielen verwendete Oberflächenspezifikationssprache hat den Arbeitsnamen "Lucia" (Language for User Centered Information Architecture). Die Namenswahl unterstreicht die Absicht, mit dieser Sprache die Oberfläche möglichst geradlinig und exakt zu beschreiben.

Lucia ist eine deskriptive, auf Beschreibung interaktiver Oberflächen ausgerichtete Sprache mit einer Syntax, die zum Zweck der Lesbarkeit an umgangssprachliches Englisch angelehnt ist.

Die Lucia-Sprache stellt eine Sammlung von oberflächentypischen Objekten bereit, welche durch Eigenschaftsattribute beschrieben und miteinander in Beziehung gesetzt werden.

Die Sprache deckt die Themenbereiche Anforderungsperspektive, Strukturperspektive, Dialogperspektive und Interaktionsperspektive sowie Schnittstellen zu funktionalen Diensten und zur Darstellungsschicht.

Bild 69: Überblick über Objekttypen in Lucia



Lucia gliedert die Oberflächenbeschreibung in drei Grundtypen von Objekten:

- **Oberflächenobjekte**
  Logische Bestandteile der Oberfläche, z.b. Ablaufschritte, Bildschirme, Elementgruppen und einzelne Ein- und Ausgabe-Elemente mit den daran geknüpften Interaktionen.
- **Kontextobjekte**
  Logische Platzhalter für die Umwelt der Benutzeroberfläche, z.B. Schnittstellen zum Datenhaushalt und zu den funktionalen Transaktionen der Anwendung (z.B. Außentemperatur), Situationen, Deklarationen von Texten und Grafiken, Schnittstellen zur grafischen Darstellung logischer Kontrollelemente.
- **Anforderungsobjekte**
  Beschreibungen von Anforderungen und Querverweise zwischen Anforderungen.

Die Syntax von Lucia legt fest, wie diese drei Grundobjekte weiter unterteilt werden und nach welchen Regeln sie einander enthalten bzw. aufeinander verweisen können.

Die Semantik von Lucia legt fest, was die Kombinationen von Objekten bedeuten und wie sie als Oberflächenspezifikation zu interpretieren sind.

Zur Abgrenzung: Lucia ist keine Programmiersprache (vgl. [Ebeling 1988, S.169-174], sondern eine auf User Interfaces zugeschnittene Spezifikationssprache, d.h. sie legt z.B. keine Implementierungsdetails fest und toleriert auch unvollständige Konstrukte.

Lucia-Beispiel: Hallo Welt

Hallo Welt

Fangen wir mit einem Hallo-Welt-Beispiel an. Wir wollen eine einfache Benutzeroberfläche beschreiben, die auf dem Bildschirm "Hallo, Welt" ausgibt und dann mit einem Knopf "Ende" beendet wird. Anhand von diesem trivialen Beispiel erklären wir die wichtigsten Sprachkonzepte und Elemente.

Später können wir unser Beispiel Schritt für Schritt um statische und dynamische Auswahllisten, Interaktion, Situationsabhängigkeit, Funktionsanforderung und Datenabfragen erweitern.

```
HCI-Model named [ Hallo_Welt_Beispiel 1 ]
!!      ( das ist der modellkopf, er identifiziert das modell als hci
spezifikation )
        ( Wir wollen eine Benutzeroberflaeche beschreiben,
     die "Hallo, Welt" ausgibt und dann mit einem Knopf "Ende"
     beendet wird. )
requirement [Anf_2]
        ( Das Beispiel soll kurz, einfach und leicht lesbar sein. )

model-revision [ 1.00 ]
!! ( die modellrevision gibt an, um welchen stand der spezifikation es
sich handelt )

lucia-revision [ 1.00 ]
!! ( die notationsrevision gibt an, mit welchem stand der
spezifikationssprache das modell erstellt wurde)

Start Step named [ Hallo_Welt ] using default Screen
        ( Im Hallo-Welt Beispiel enthaelt unsere Oberflaeche nur
     einen einzigen logischen Schritt, welcher als IO-Medium
     den Bildschirm gleichen Namens nutzt. Der logische Schritt selbst
     ist in unserem Beispiel nicht als Kontrollinstrument aktiv.
     Weitere Einsatzmoeglichkeiten der logischen Schritte
     werden in anderen Beispielen diskutiert, siehe ~[Anf_2]. )

requirement ( Der Bildschirm dient als Ein- und Ausgabemedium fuer
     den logischen Schritt gleichen Namens, in den er ablaufmaessig
     eingebettet ist. )

Screen named [ Hallo_Welt ]
  using Layout ~[ Hallo_Anordnung ]
   contains
   {
   Text [ Hallo, Welt! ]
     placed at ~[titelzeile]
     using Widget ~[spezielles_text_widget]
   Button [ Ende ]
     placed at ~[fusszeile]
```

```
     using default Widget
     on command unconditionally leave current Step
   }

Layout named [ Hallo_Anordnung ]
!! ( Html ist nur ein Beispiel fuer die Formulierung eines Layouts.
     Hier koennte eine beliebige andere Layoutvorschrift, die
     der Spezifikationsleser oder ein Interpreter versteht, stehen.
     Zum Umfang von Konkreto gehoeren lediglich die Platzhalter. )
  {::
  <html><body><table>
    <tr><td>placeholder named [titelzeile]</td></tr>
    <tr><td>placeholder named [koerper]</td></tr>
    <tr><td>placeholder named [fusszeile]</td></tr>
  </table></body></html>
  ::}

Widget named [spezielles_text_widget]
!! ( Hier ist die Schnittstelle zu Darstellungselementen.
     Man kann hier, umgangssprachlich oder formalisiert, z.B. in SVG
     oder in HTML spezifizieren, wie das Widget zur Darstellung
     des logischen Bedienungselement aussehen soll. )
   {::
   <h1>~[value]</h1>
   ::}

  Functionality [ anwendungsfunktion_1 ]
  !! ( thats the interface to the business logic )
        {::
        some code to call the functionality in its runtime environment
        can also be used to call a stub
        ::}
```

Sehen wir uns das Beispiel näher an: Es zeigt, wie die Verquickung von Abläufen, Inhalten, Layout, Darstellung, Interaktionen und Anforderungen systematisch aufgegliedert und zu einem abgestimmten Ganzen zusammengefügt wird.

Nachfolgend werden anhand der im Hallo-Welt-Beispiel verwendeten Ausdruckformen die von der Lucia-Sprache gelieferten Lösungen zur Formulierung von komplexen Zusammenhängen und Abhängigkeiten zwischen den unterschiedlichen Bausteinen der Oberfläche skizziert.

HCI-Model named [Hallo_Welt_Beispiel]

sagt aus, dass unser Spezifikationsmodell den Namen "Hallo_Welt_Beispiel" hat. Namensdefinitionen werden mit dem Wort "**named**" eingeleitet. Statt der Anführungszeichen, die wir oft in Fließtexten der Spezifikation benötigen, werden eckige Klammern **[ ]** als Begrenzer für Namen und Inhalte verwendet.

In einer Spezifikation kommen viele Namen und Querverweise vor. Deshalb wird zur besseren Unterscheidung eine Namensfestlegung mit "**named [name]**" notiert und eine Namensreferenz

mit "**~[name]**". Die in einer Lucia-Spezifikation verwendeten Namen dürfen beliebige Zeichen außer eckigen Klammern, Punkt, Komma und Anführungszeichen (**[ ] . , "**), also auch Leerzeichen und Umlaute enthalten.

!! [Wir wollen …]

legt eine Anforderung fest.

!! leitet eine Anforderungsbeschreibung ein. Eine Anforderung kann, muss aber nicht, einen sie identifizierenden Namen haben. „**!! named [Anf_2] [Das Beispiel soll …]**“ legt eine Anforderung mit dem Namen "Anf_2" fest. Die Beschreibung der Anforderung folgt in einem ebenfalls mit **[** und **]** begrenzten Block. Innerhalb der Anforderungsbeschreibung darf mit **~[name]** auf andere Anforderungen und auch auf beliebige andere benannte Elemente der Spezifikation quer verwiesen werden.

Eine Anforderung gehört zu dem nächsten über ihr befindlichen Modellelement; im Hallo-Welt-Beispiel beziehen sich die ersten beiden Anforderungen also auf das Modellelement **HCI-Model** und damit auf die gesamte Oberflächenspezifikation. Die unbenannte Anforderung **„!! [Im Hallo-Welt Beispiel enthält …]**“ bezieht sich auf „**Step named [Hallo_Welt]**“.

Eine Anforderung muss keinen Namen haben, allerdings kann dann von anderen Stellen der Spezifikation oder auch bei Auswertungen nicht Bezug auf so eine unbenannte Anforderung genommen werden. Daher empfiehlt es sich, „echte“ Anforderungen immer mit einem Namen zu versehen.

Eine Anforderungsbeschreibung kann bei Bedarf weiter strukturiert werden, d.h. z.B. in Ausgangssituation, Zielsetzung, Vor- und Nachbedingungen. Wann und wie diese Möglichkeit angewandt wird, zeige ich in einem späteren Beispiel, siehe ~[Anf_2] :-).

Start Step named [Hallo_Welt] using default Screen

definiert den ersten Ablaufschritt im Workflow der Benutzeroberfläche. Die Wortfolge „**Start Step**“ zeichnet den Schritt als den ersten Schritt im logischen Ablauf der Oberfläche aus.

Da unser Beispiel nur einen logischen Ablaufschritt hat, würde das Auslassen von **Start** keine Auswirkungen haben. Wenn „**Start Step**“ nicht angegeben wird, gilt die Konvention, dass der Ablauf mit dem in Definitionsreihenfolge ersten Schritt des Modells beginnt.

Höchstens ein **Step** des Modells darf mit dem Prädikat **Start** versehen werden. Typischerweise ist dieser Ablaufschritt ein

Login oder ein Hauptmenü. Der Startschritt übernimmt das Hochfahren der Oberfläche und kontrolliert den anfänglichen Übergang zu anderen Ablaufschritten.

Mit „**using default Screen**“ wird festgelegt, dass der logische Schritt „Hallo_Welt“ als Darstellungs- und Eingabemedium einen Bildschirm gleichen Namens nutzt.

Schritte können hierarchisch angeordnet werden und bieten vielfältige Konstrukte zur Ablaufspezifikation mit Unterstützung der orthogonalen Verwendung von hierarchischer und prototypischer Vererbung, die unser einfaches Beispiel jedoch nicht demonstriert.

Screen named [Hallo_Welt] …

beschreibt nun die Inhalte der Bildschirmmaske, die der Anwender sieht, und die Interaktionsmöglichkeiten auf dieser Bildschirmmaske.

Mit „**using Layout ~[Hallo_Anordnung]**“ wird dem Bildschirm ein Anordnungsschema zugeordnet, auf das die Dialogelemente Bezug nehmen können.

Innerhalb von „**contains […]**“ werden die Inhalte des Screens, also Ein- und Ausgabeelemente mit ihren Eigenschaften beschrieben.

Text und **Button** sind logische Dialogelemente. Sie sind auf dieser Ebene noch unabhängig von ihrer tatsächlichen Darstellung (Aussehen) auf der Oberfläche, legen aber fest, was und wie zwischen Anwender und Anwendung kommuniziert wird. Weitere typische logische Dialogelemente, auf die in diesem Beispiel nicht näher eingegangen wird, sind Selector (z.B. Listbox, Combobox, Menü, Checkbox, Radiobutton, Slider), Editfields in verschiedenen Spezialisierungen und Kompositionen aus diesen Grundtypen.

Allen logischen Dialogelementen ist gemeinsam, dass sie Informationen zur und von der Darstellungsebene hin- und her transportieren. Der Informationsgehalt ist dabei abhängig vom logischen Typ des Dialogelements (z.B. dient eine Auswahlliste zur Auswahl aus den angebotenen Möglichkeiten; ein Button hingegen zum Aktivieren einer bestimmten Möglichkeit), jedoch unabhängig vom Look-And-Feel auf der Darstellungsebene (z.B. ob die Auswahl über RadioButtons oder über eine DropDown-Listbox getroffen wurde, ist für die logische Ebene unwesentlich).

Logische Dialogelemente sind mit einer Reihe von Eigenschaften ausgestattet, mit denen ihnen Ein- und Ausgabeinhalte, Situationen und Interaktionen, auf denen sie operieren, zugewiesen werden. Die Schnittstelle des logischen Dialogelements zur Darstellungsebene wird ebenfalls durch Eigenschaften ausgedrückt.

In unserem Hallo-Welt-Beispiel benutzen wir zur Veranschaulichung der Screenbeschreibung zwei logische Dialogelemente (Text und Button) mit wenigen Eigenschaften.

Text [Hallo, Welt!]
placed at ~[titelzeile]
using Widget ~[spezielles_text_widget]

legt fest, dass der Text „Hallo, Welt!“ an der Layoutposition „titelzeile“ ausgegeben werden soll. Zur Darstellung des Texts soll dabei die im Widget „spezielles_text_widget“ festgelegte Renderingvorschrift verwendet werden.

Button [Ende]
placed at ~[fusszeile]
using default Widget
on command unconditionally leave current Step

legt fest, dass der mit „Ende“ beschriftete, unbenannte Command Button an der Layoutposition „fusszeile“ angezeigt werden soll. Zum Rendering des Buttons soll dabei ein generisches Widget (z.B. der von Windows bereitgestellte Pushbutton) verwendet werden. Wenn der Button geklickt wird, soll der Screen und der Step „Hallo_Welt“ und damit die Anwendung verlassen werden. Beim Drücken des Buttons sollen keine Bedingungen geprüft und keine Rückfragen (etwa "Sind Sie sicher") gestellt werden.

Die vielfältigen Möglichkeiten der Situationsprüfung und der Reaktion auf Eingaben und Kommandos werden anhand von anderen Beispielen verdeutlicht.

Layout named [Hallo_Anordnung] […]

leitet die Definition eines Layouts ein. Das Layout beschreibt eine Schnittstelle zur Darstellungsebene der Oberfläche. Im Layout können die statischen Anteile von Screens beschrieben und der Screen in Bereiche unterteilt werden.

Die Beschreibung des Layouts kann in einem beliebigen Format erfolgen und ist von Lucia nicht definiert. Zur Lucia-Sprache gehören lediglich die Platzhalter, auf die sich Dialogelemente zur Positionierung mit "**placed at ~[platzhaltername]**" beziehen können. Ein Positionierungsplatzhalter wird mit "**placeholder named [platzhaltername]**" festgelegt.

Zur Festlegung von Layouts bietet sich insbesondere HTML und SVG an. Zwischen Auszeichnungselementen können die Platzhalter eingestreut werden, an welchen die Dialogelemente eingesetzt werden können. Dialogelemente, die keinen Bezug auf Layoutplatzhalter nehmen, landen im Bereich "**placeholder named [content]**", wenn das Layout einen solchen Platzhalter festlegt.

Benutzt der Screen kein Layout, dann werden die Elemente in Definitionsreihenfolge platziert. Hat das vom Screen benutzte layout keinen [content]-Platzhalter, dann werden die Dialogelemente nach den Layoutelementen auf dem Bildschirm ausgegeben.

Zur Festlegung der grafischen Eigenschaften von Widgets bietet sich, wie schon für Layouts, XHTML und SVG an. Zwischen den Auszeichnungselementen, die das Widget rendern, können die Platzhalter eingestreut werden, an welchen die Eigenschaften von Dialogelementen eingesetzt werden können.

Widget named [spezielles_text_widget] […]

leitet die Definition eines Widgets ein. Das Widget beschreibt, wie das Layout, eine Schnittstelle zur Darstellungsebene der Oberfläche. Im Widget kann das Aussehen und die interne Präsentationslogik (z.B. wie sich das Widget beim Fokussieren verändert) des Darstellungselements beschrieben werden.

Die Beschreibung des Widgets kann in einem beliebigen Format erfolgen und ist von Lucia nicht definiert. Zur Lucia-Sprache gehören lediglich die Platzhalter für den Inhalt "**[value]**" und weitere Eigenschaftswerte von logischen Dialogelementen, auf die unser Beispiel jedoch nicht eingeht.

Das Beispiel zeigt nur eine kleine Auswahl der Sprachkonstrukte der Oberflächenspezifikationssprache Lucia.

Es sollte Ihnen jedoch in Form eines vertikalen Durchstichs einen Eindruck von den unterschiedlichen Themen (Sichten, Perspektiven, Bestandteilen), die in einer Spezifikation beschrieben und in Bezug zueinander gesetzt werden müssen, verschafft haben.

6.6 Umfang der Lucia-Sprache

Übersicht der Ausdrucksmöglichkeiten für UI-Sachverhalte der Lucia Sprache. Die Schlüsselworte der Sprache repräsentieren wichtigste Elemente der Oberfläche bzw. ihrer Beschreibung.

Anforderungsperspektive

- Anforderung: `REQUIREMENT`
  - Anforderungen identifizieren
  - Anforderung beschreiben
  - Bezug zu anderen Modellelementen herstellen
  - Querverweise

Strukturperspektive

- Ablaufschritt: `STEP`
  - Aneinanderreihung
  - Schachtelung
  - Stepeigenschaften (z.B. Modalität)
  - Schritte fallabhängig durchführen
  - Schritte parallel durchführen
  - Hierarchische Vererbung
  - Vererbung aus Musterschritten

Dialogperspektive

- Dialog: `SCREEN / DIALOG`
  - Bezug zum logischen Schritt
  - Bezug zu einer Layoutvorschrift
- Widgetgruppe
  - Platzierung
  - Darstellungsbezug / Rendering
- Widget
  - Platzierung
  - Darstellungsbezug / Rendering
  - Kontrollelement
  - Datenbezug
  - Eingabeschablonen
  - Sichtbarkeit / Modus (z.B. enabled/disabled)

Interaktionsperspektive

- Interaktion
  - Bezug zum logischen Schritt

  - Bezug zum Dialogelement
  - Bezug zu Situationen (Prozesslogik)
  - Auslösen von Funktionen
  - Manipulieren der Eigenschaften von Schritten und Dialogelementen

Kontextperspektive

- Situation / Guard
- Datenverwendung / Application Value
- Funktionsverwendung / Application Function
- Widget-Realisierung
- Layoutvorschrift
- Laufzeitkontext
  - Current Step Stack
  - Interaktionshistorie
  - Fokus Vektor
  - Stephistorie

Modellperspektive

- Metainformationen über das Modell
  - Modellierungsstand (für das gesamte Modell und herunter gebrochen auf einzelne Modellelemente)
  - Inkludierungen
  - Version
  - Autor

Keine Steuerungsvariablen

Lucia bietet bewusst keine Variablen an. Die UI-Session ist selbst ein AnwendungsObjekt. Um Werte abzurufen oder zu speichern gibt es zwei Möglichkeiten: ***Application Values*** und ***unsichtbare Kontrollelemente*** (z.B. versteckte Textfelder).

Das nachfolgende Blockdiagramm zeigt eine Übersicht der Sprachelemente von Lucia.

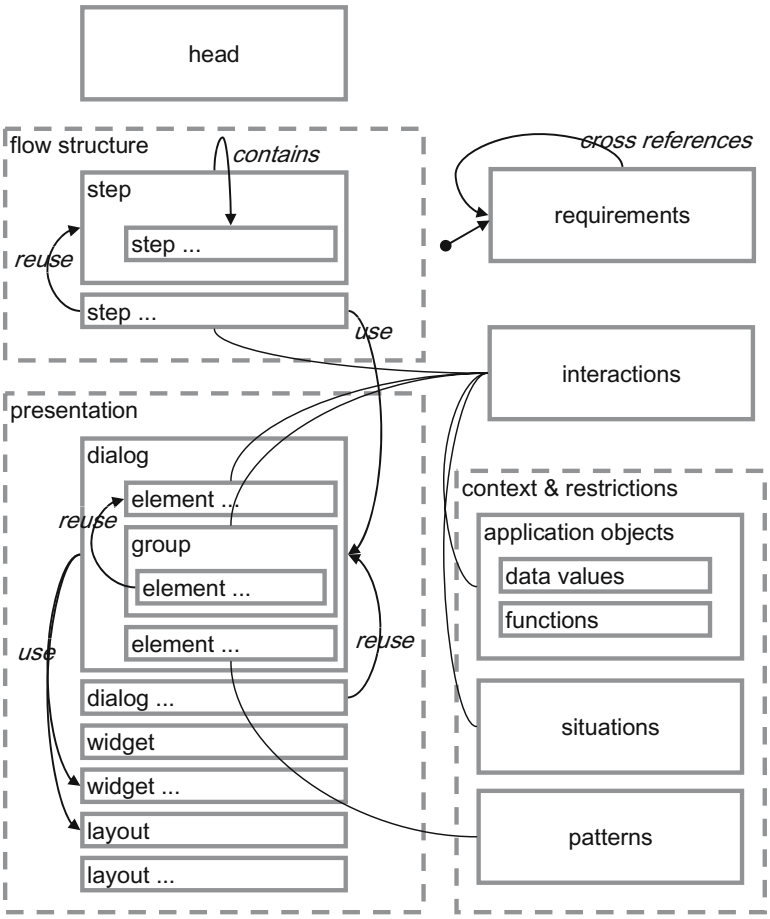


Bild 70: Sprachstruktur von Lucia

6.7 Lucia-Grammatik

Die nachfolgende Parsergrammatik in ANTLR definiert die Syntax der Lucia-Sprache (Ausschnitt). Die Sprache ermöglicht eine eindeutige Beschreibung von User Interfaces und unterstützt die Abbildung der UI-Perspektiven in einer User Interface Spezifikation.

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// modellgerüst
// .. beschreibt den aufbau des hci modells
// .. definiert die reihenfolge der modellteile: modellkopf, steps,
screens, layouts, widgets

lucia:
        modelhead_
        (step_)+
        (screen_)*
        (layout_)*
```

Modellstruktur

```
        (widget_)*
        (context_)*
        ;


/// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// modellkopf
// .. identifiziert die datei als hci model
// .. legt den namen des modells fest
// .. liefert versionsinformationen zum modell und zur modellnotation
// .. legt gegebenenfalls modelglobale interaktionen fest
// .. legt gegebenenfalls modelglobale anforderungen fest
```

Verwaltungs-Informationen

```
modelhead_:
        model_name_
        model_revision_
        lucia_revision_
        ( interaction_ )*
        ;

model_name_:
// .. legt den namen des modells fest
// .. legt eventuelle bezugnahmen auf modelle in anderen systemen (
alias namen)
        MODEL name_definition_ (requirements_)?
        ;

model_revision_:
// .. legt fest, mit welchem versionsstand das modell bezeichnet ist
// .. die angabe der modellversion ist optional
        (
        MODEL_REVISION name_definition_ (requirements_)?
        |
        )
        ;

lucia_revision_:
// .. nennt die sprachversion, in der das modell verfasst wurde
// .. die angabe der konkreto-sprachversion ist erforderlich
(pflichtangabe)
        LUCIA_REVISION name_definition_ (requirements_)?
        ;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// ablaufschritte der benutzeroberfläche
// .. beschreiben die ablaufstruktur der oberfläche
// .. können ineinander geschachtelt werden
```

Ablaufstruktur

```
step_:
        STEP
        (name_definition_ )?
        (requirements_)?
        (using_screen_clause_ (requirements_)?)?
        (based_on_clause_ (requirements_)?)?
        (step_content_ (requirements_)?)?
        ;

step_content_:
```

```
        (CONTAINS)? OPENBLOCK
        (requirements_    )?
        (
        (step_)*
        )
        CLOSEBLOCK
        ;

screen_:                                                    Dialogsicht
        SCREEN
        (name_definition_)?
        (requirements_)?
        (using_layout_clause_ (requirements_)?)?
        (interactions_)?
        (screen_content_ (requirements_)? (interactions_)? )?
        ;

layout_:
        LAYOUT
        (name_definition_)?
        (requirements_)? taggedcode_
        ;

widget_:
        WIDGET
        (name_definition_)?
        (requirements_)? taggedcode_
        ;

context_:                                                   Kontextsicht
        FUNCTIONALITY
        (name_definition_)?
        (requirements_)? taggedcode_
        ;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// bildschirme der benutzeroberfläche
// .. sind der io-kanal zwischen benutzer und hci
// .. enthalten ein- und ausgabeelemente

screen_content_:
        (CONTAINS)? OPENBLOCK
        (requirements_    )?
        (
        (container_ | element_)*
        )
        CLOSEBLOCK
        ;

using_screen_clause_:
// io-kanal bildschirm
// .. steps können (müssen aber nicht) einen screen verwenden (über
diesen screen erfolgt die io)
// .. falls screen nicht angegeben, dann wird angenommen, dass der zum
step zugehörige screen den selben namen hat
// .. "using default" und keine angabe sind aquivalent
// .. ist dieser default-screen nicht defniert, dann wird angenommen,
dass der step keinen bildschirm verwendet
```

```
// .. ist explizit "using screen ~[id]" angegeben und der screen mit
dieser id nicht definiert, dann wird das als spezifikationsfehler
gewertet

        USING
                        ( SCREEN name_reference_
                        | DEFAULT
                        )
        ;

using_layout_clause_:
// .. screens können (müssen aber nicht) ein layout verwenden
// .. das layout ist eine schablone für die positionierung der inhalte
auf dem screen
        USING
                        ( LAYOUT name_reference_
                        | DEFAULT
                        )
        ;


based_on_clause_:
// .. steps können (müssen aber nicht) auf anderen steps aufbauen
// .. so ein step erbt die eigenschaften des mit based on
referenzierten step
        BASED_ON name_reference_
        ;
```

Logische Kontrollelemente

```
container_:
        CONTAINER (name_definition_)?
        OPENBLOCK CLOSEBLOCK
        ;

element_:
        ( display_element_ | button_element_ | item_list_ |
checkbox_element_ )
        ( name_definition_ )?
        element_commons_
        (interactions_)?
        ;

element_commons_:
        (
        PLACED_AT name_reference_
        using_widget_clause_
        )?
        ;

display_element_:
        DISPLAY
        ;

button_element_:
        BUTTON
        ;

item_list_:
        ITEMLIST
        ;
```

```
checkbox_element_:
         CHECKBOX
         ;

using_widget_clause_:
// .. elemente und containet können (müssen aber nicht) ein widget
verwenden
// .. das widget ist eine schablone für das rendering des elements auf
dem screen
// .. bei detaillierung der spezifikation sollten elementen nach und
nach widgets zugeordnet werden

         USING
                         ( WIDGET name_reference_
                         | DEFAULT
                         )
         ;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Definitionen und Verweise
// .. definiert wird ein name oder eine zeichenkette
// .. namen wird vorzug gegeben (d.h. definitionen werden als name
aufgefasst, wenn sie die kriterien für eine id erfüllen)
// Eine Namensdefinition kann implizit (da an dieser Stelle erwartet)
oder explizit durch das Schlüsselwort
// 'named' eingeleitet werden. In bestimmten Fällen, z.B. bei einer
Anforderung, ist eine explizite
// Namenseinleitung erforderlich, weil sonst der Anforderungstext mit
seiner [optional vergebbaren]
// Benamung verwechselt werden könnte.

name_definition_:                                          Namensdefinition
                  (NAMED)? x:NAME_DEFINITION ;

name_reference_:                                           Namensverwendung
                  x:NAME_REFERENCE;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// interaktionen
// .. werden von ereignissen ausgelöst
// .. definieren reaktionen auf die ereignisse
// .. können bedingungen vor die reaktionen stellen

interactions_:                                             Interaktionen
         (interaction_)+
         ;

interaction_:
         (
         onEnterInteraction_ | onSignalInteraction_ |
onLeaveInteraction_ | onCommandInteraction_
         )
         ;

// model_interaction_:
// .. legt modellglobale interaktionen fest
```

```
// .. interaktionen, die beim start des hci ausgeführt werden
(onEnter)
// .. interaktionen, die ereignisgesteuert ausgeführt werden
(onSignal)

// screen_interaction_:
// .. legt globale screen interaktionen fest
// .. interaktionen, die beim start des screens ausgeführt werden
(onEnter)
// .. interaktionen, die ereignisgesteuert ausgeführt werden
(onSignal)

//element_interaction_:
// .. legt elementspezifische interaktionen fest
//       ( common_interaction_ | onCommandInteraction_ )

// common_interaction_:
//       onEnterInteraction_ | onSignalInteraction_ |
onLeaveInteraction_

// .. beim betreten des modells, eines steps, eines containers, eines
elements
// .. ein step wird immer vor dem dazugehörigen screen betreten
// .. die step-on-enters werden unmittelbar nach erlangen der
kontrolle durch den step ausgeführt
// .. die screen-on-enters werden nach ausgabe [bzw. anstossen der
ausgabe] der screeninhalte ausgeführt
// .. die element-on-enters werden nach dem abschluss des screen-
aufbaus bei übergabe der kontrolle an einzelne elemente ausgeführt
onEnterInteraction_:
         ON_ENTER (requirements_)?
         (condition_)? action_
         ;

// .. beim verlassen (siehe onEnter)
onLeaveInteraction_:
         ON_LEAVE (requirements_)?
         (condition_)? action_
         ;

// .. bei Auftreten eines Signals
onSignalInteraction_:
         ON_SIGNAL (requirements_)?
         (condition_)? action_
         ;

onCommandInteraction_:
         ON_COMMAND (requirements_)?
         (condition_)? action_
         ;
```

Bedingungen

```
condition_:
         (
         UNCONDITIONALLY
         | WHEN expression_
         )
         (requirements_)?
         ;
```

```
expression_:
        (
        name_reference_ EQUALS name_reference_
        )
        ;

action_:
        (
        x:LEAVE
        | y:PASS
        | z:APPLY name_reference_
        | v:NEXTSTEP (name_reference_)?
    )
        (requirements_)?
    ;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// business logic context
// .. funktionalitäten, die die anwendung zur verfügung stellt
// .. es reicht der name der funktionalität, es geht darum zu wissen,
dass was an einer stelle des hci getan wird, nicht wie es getan wird
// .. für das hci relevante antworten der funktionalität (nicht daten,
sondern status- und fehlercodes, z.b. "sim-nicht-erkannt")
// .. die mit der funktionalität assozierten daten (können natürlich
auch statuscodes enthalten, die wahl was wo erwartet wird, ist eine
designentscheidung)



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// requirements - anforderungen
// .. sind benannt oder unbenannt
// .. auf die benannten anforderungen kann man verweisen
// .. die unbenannten anforderungen haben kommentar-charakter
// .. anforderungen können auf andere modellstellen und aufeinander
verweisen
// .. andere modellstellen können auf anforderungen verweisen
// .. anforderungen legitimieren funktionale modellelemente
// .. jedes funktionale modellelment sollte auf mindestens eine
anforderung verweisen

requirements_:                                          Anforderungen
        (requirement_)+                                 Querverweise
        ;

requirement_:
        (
        REQUIREMENT
        x:INSIDE_REQUIREMENT
        ;


taggedcode_:
        x:TAGGEDCODE
        ;
```

Einige Semantikregeln

Einschub: Reihenfolge des Abarbeitens einer UI-Beschreibung; Zusammenarbeit zwischen Steps und Screens.

- 1: Betreten der Hülle des Steps
- 1b: Abarbeiten der on-enter-Ereignisse des Steps
- 2: Wenn der Step einen Screen benutzt, dann Screen aufbauen
- 3: Abarbeiten der on-enter-Ereignisse des Screens (Dialogs)
- 4: Hauptschleife:
  - on-Signal-Ereignisse und
  - andere Ereignisse verarbeiten
  - in der Reihenfolge Control-Dialog-Step entlang der Vererbungshierarchie der Steps
- 5: Abarbeiten der on-leave-Ereignisse des Screens
- 6: Abarbeiten der on-leave-Ereignisse des Steps

Regeln:

- In einer Step-Schachtelung werden die geschachtelten Steps in Definitionsreihenfolge betreten.
- Es werden zuerst die on-Enter-Interaktionen ausgeführt.
- Dann wird der Screen oder der VoiceDialog ausgeführt, falls der Step mit einem solchen assoziiert ist.

Zum Verständnis von „on Signal“:

- Die Anwendung setzt immer dann ein Signal an die Oberfläche ab, wenn es für die Oberfläche etwas zu tun gibt.
- Das Signal wird dann vom aktiven Punkt der Oberfläche gewürdigt (also es werden Situationen geprüft und gegebenenfalls Reaktionen eingeleitet).
- Das passiert in der Reihenfolge
  - Element,
  - Elementgruppe,
  - Screen,
  - Step-Hülle,
  - Hierarchie der semiaktiven Steps.
- Auf Anforderung kann jede Signal verarbeitende Stelle das Signal „aufessen“, dann wird es nicht weiter nach oben propagiert.

Zum Verständnis von „on Situation“:

- Das Situation-Ereignis wird immer dann abgesetzt, wenn sich eine Situation ändert.
- Optimalerweise wird in der UI-Ablaufmaschine nicht nach Situationen gepollt (also nicht ständig überprüft, ob sich vielleicht eine Situation verändert hat), sondern der Kontexthandler setzt das Situation-Changed-Signal im aktuellen Kontext ab.
- Auf diese Weise passiert im User Interface kein ständiges Überprüfen von Situationsänderungen, die im aktuellen Ablaufpunkt des UI gar nicht relevant sind.

Zum Verständnis der Value-Zuweisung:

- Wenn ein Kontrollelement seinen Anzeigewert aus einem Application Value bezieht, wird impliziert, dass die Anzeige des Controls jedes Mal aktualisiert wird, wenn sich der Application Value ändert.

Zum Verständnis des Zusammenhangs zwischen Signal, Application Value und Situation:

- Wenn das UI ein Signal erhält, muss es das Signal in der Regel vor einer Reaktion qualifizieren, d.h. eine Situation oder einen Anwendungswert auswerten.
- Die direkte Kopplung von Kontrollelementen an Situationen und Anwendungswerte impliziert ein zweckgebundenes Signal an das Kontrollelement, wenn sich die Situation oder der Datenwert ändert.

Beispiel: Telefonbedienung

Ein Modell für das User Interface des Telefonmoduls in einem Fahrzeug-Infotainmentsystem.

Das nachstehende Beispiel zeigt, wie Gruppen aus logischen Schritten gebildet werden und wie einzelne logische Schritte mit Bildschirmen unterlegt werden.

```
hci model [ phone ]
!!      (
        Telefonmodul im Fahrzeug-Infotainmentsystem.

        Die grobe Beschreibung des Use Cases ist:
```

```
        - Telefon mit Sim-Karte in Betrieb nehmen.
        - PIN und PUK Eingabe.
        - Rufnummer eingeben und anrufen.
        - Eingehenden Anruf annehmen aus beliebigem Menü heraus.
        - Anruf beenden.
        - Adressbuch mit statischen Einträgen.
        - Umschalten zwischen den Bereichsmenüs Navigation, Radio,
Telefon, Klima und Einstellungen.
        )
lucia-revision [1.02]
!!      ( Erstellt mit LUCIA Notation 1.02 )

step [hauptmenu]
!!      (
        aus dem hauptmenü
        kann man in die einzelnen funktionsbereiche wechseln
        )

step [phonemenu]
!!      (
        The communication menu enables the user to use
telecommunication services
        and other car services (depending on the car configuration) in
order to exchange information.
        )

step-group [phone]
        !!      ( das telefonmodul stellt die telefonfunktionalität
zur verfügung )
        {
        !!      ( die funktionalität ist hierarchisch gegliedert,
beim ablauf ist aber immer nur ein step aktiv )
        step-group [sim-card-checken]
                {
                step [sim-card-einlegen-bitte]
                step [sim-card-ok]
                step [pin-eingeben]
                step [puk-eingeben]
                step [ende-gelände]
                }
        step-group [anrufen]
                {
                step [aus-adressbuch-anrufen] based on ~[adressbuch]
                step [rufnummer-wählen-und-anrufen]
                }
        step-group [telefonieren]
                {
                step [eingehender-anruf]
                step [eingehender-anruf-annehmen]
                step [auflegen]
                step [eingehender-anruf-ablehnen] based on
~[auflegen]
                step [gespräch-läuft]
                }
        step [adressbuch]
        }

screen [sim-card-checken]
```

```
!! ( der bildschirm, der während des steps ~[sim-card-checken]
eingeblendet wird )

         on enter apply ~[sim-card-checken-funktion]
!! (Es soll erst der Hinweis ausgegeben und dann die Funktion
aufgerufen werden. Der Hinweis soll solange auf dem Bildschirm stehen,
solange die Funktion durchgeführt wird. In Abhängigkeit vom Ergebnis
der Funktion soll der Anwender zum Einlegen der sim-Card aufgefordert,
zum Eingeben der Pin aufgefordert oder zum Telefonmenu weitergeleitet
werden.)

         on signal
!! (nach dem checken der simcard ergebnisabhängig verfahren )
         when ~[abfragen-ob-sim-card-checken-funktion.response] =
                  ~[simcard_needs_pin]
         nextstep ~[sim-card.checken.pin-eingeben]

         !! ( on leave apply ~[abfragen-ob-sim-card-checken-
funktion.response=simcard_ready_to_use] !! (heisst pin wurde bereits
eingegeben) )
         {
         display [ simcard wird gecheckt ... ]
         display [ gleich gehts weiter. ]
         display [ noch mehr ]
         }


functionality [sim-card-checken-funktion]
         {::
         application-value [beispiel]
         response [beispiel2]
         ::}
```

Beispiel: Wertebereiche und Auswahllisten

Wertebereiche festlegen

Festlegen von Wertebereich-Mustern, z.B. für ein Eingabefeld zur Betragsangabe.

```
pattern named [betrag]
  values 0..10, 1000, 1500, 2000..2010, 2500, 3500
  format z.zz9,99


pattern named [blz]
  values 0..9999999
  format 999b999b99
```

Die Angabe `values` legt den gültigen Wertebereich für einen Eingabewert (z.B. eine Zahl oder einen Schlüssel) fest.

Die Formatangabe `format` legt z.B. die Anzeigeaufbereitung von Zahlen fest. Der Format-Ausdruck ist an die Notation der Cobol-Picture-Klausel angelehnt.

9 steht für eine Dezimalziffer.

z steht für eine Dezimalziffer, die nicht angezeigt werden soll, wenn sie den Wert Null hat. Damit lassen sich z.B. führende Nullen unterdrücken.

b steht für „blank“ und erzeugt ein Leerzeichen. Damit lassen sich z.B. bei einer Bankleitzahl zur besseren Lesbarkeit Leerzeichen einfügen.

Der Dezimalpunkt, das Komma und andere Interpunktionssymbole werden wie das Leerzeichen an der Stelle eingefügt, an der sie in der Formatangabe festgelegt wurden.

Wertebereiche verwenden

Verwenden von festgelegten Wertebereich-Mustern:

```
editfield
  using pattern ~betrag
editfield
  using pattern ~blz
selectfield
  using pattern ~betrag
```

Hinweis zum Referenzieren von Modellbestandteilen: Das Verweisen auf benannte Einheiten des UI-Modells erfolgt in Lucia durch `[~name]`. Verschiedene Modellteile (Steps, Layouts, Patterns) können den gleichen (lokalen) Namen haben. Um die Verweise auf solche Namen eindeutig zu machen, kann dem Namen der Entitätstyp bzw. die Namenshierarchie vorangestellt werden. Zum Beispiel: `use ~pattern.blz` ist gleichwertig mit `use ~blz`. Anderes Beispiel: `go to step ~auswahl` ist gleichwertig mit `go to step ~menues.hauptmenue.auswahl`, sofern das der im Modell nächstgelegene logische Schritt ist.

Ein und dasselbe Wertebereich-Muster kann von verschiedenen Kontrollelementen verwendet werden. Die Art der zur Verfügung stehenden Eingabemöglichkeiten hängt von dem logischen Kontrollelement und von der physischen Darstellung des Widgets, das verwendet wird, ab.

Zum Beispiel bedeutet

```
selectfield
  using pattern ~betrag
  using widget ~dropdownlist
```

dass der Wertebereich von `betrag` als Auswahlfeld in einer aufklappbaren Auswahlliste („Combo-Box“ bzw. „Drop-Down-List“) bereitgestellt wird. Dabei wird jeder zulässige Wert aus dem Wertebereich von `betrag` als Eintrag in der Auswahlliste bereitgestellt.

Bei Zahlen ist die Schrittweite der Auswahl davon abhängig, wie viele Nachkommastellen die Formatangabe festlegt.

Bei einer Bereitstellung als Auswahlliste enthält die durch

```
pattern named [betrag]
  values 0..10, 1000, 1500, 2000..2010, 2500, 3500
  format z.zz9,99
```

festgelegte Auswahlliste also die folgenden Werte: 0,00; 0,01, ... 9,98; 9,99, 10,00; 1.000,00; 1.500,00; 2.000,00; ...; 2009,99; 2.010,00; 2.500,00; 3.500,00.

Weitere Wertebereichs-Muster können sein: Dynamische Listen, Kombinationen aus festen Werten mit dynamischen Anteilen sowie direkte Reaktionen auf die Auswahl einzelner Werte.

Zum Beispiel:

```
pattern named [feiertage]
  {
  values „1. Januar - Neujjahr“, „25. Dezember - Weihnachten“
  values from function ~feiertagsliste
  }
  on select set ~kontrollanzeige.value to ~.value
```

Ein Pattern definiert Wertebereiche für Eingaben (und Ausgaben)

- durch reguläre Ausdrücke
- Aufzählung von Einzelwerten
- Dynamische Wertebereiche
- Ausschluss von Einzelwerten und dynamischen Wertebereichen

An Wertegruppen und einzelne Werte können Reaktionen - zum Beispiel Fehlermeldungen (insbesondere auf Ausschlusswerte und Werte außerhalb des Wertebereichs), Value-Checker, Ergänzer (automatisches Ergänzen der Eingaben anhand von Anfangsbuchstabenfolge) - geknüpft werden.

Formatschablonen ermöglichen die Präsentationsaufbereitung von numerischen und alphanumerischen Werten.

Üblicherweise werden Pattern-Eigenschaften in einem Kontextobjekt zusammengefasst und können mittels `using ~name` beliebig oft verwendet werden. Bei Bedarf können Pattern-Eigenschaften auch direkt in der Definition eines Kontrollelements festgelegt werden.

Zum Beispiel:

```
selectionfield named [monatsauswahl]
  using widget ~auswahlkreis
  pattern values „Januar“, „Februar“, „März“, ... , „Dezember
```

6.8 Nutzen

Der Nutzen einer eindeutigen UI-Beschreibungssprache ist: Die unterschiedlichen Vorstellungswelten der Interessengruppen können ohne Zusatzaufwand zu einem Gesamtbild vereinigt werden. Das Gesamtbild ist eindeutig und kann maschinell ausgewertet und transformiert werden.

Die formale UI-Beschreibung liefert etwas, was aus PowerPoint-Folien und aus formfreien Beschreibungen nicht herausgelesen werden kann: Hinreichend genaue, eindeutige Informationen über geforderte Eigenschaften der Oberfläche.

Der Nutzen der Lucia-Sprache lässt sich anhand von Touristenenglisch und der Eurowährung bildhaft erklären. Man versteht das bereiste Land nicht ganz, hat vielleicht einen eingeschränkten Wortschatz, aber man hat eine ganz gute Vorstellung davon, was ein Euro ist und fühlt sich daher einigermaßen sicher bei Kaufentscheidungen.

Die unterschiedlichen Vorstellungen und Erfahrungswelten der Anwendervertreter im Land der Entwickler und umgekehrt finden im standardisierten Wortschatz eine gemeinsame Basis und ermöglichen eine schnellere Vertiefung in die Belange des Anderen beim Erreichen eines gemeinsamen Ziels.

Das Ziel ist eine funktionierende Anwendung, die an der Oberfläche für die Anwender in ihrem Sinne verwendet werden kann und in der Technik so umsetzbar ist, wie es von den Ingenieuren vertreten werden kann.

Die Sprache ist die Wohnung des Seins

Ich füge als Motivation für das Verwenden einer Sprache zu UI-Modellierung hinzu: Was man in einer Sprache für andere verständlich und eindeutig ausdrücken kann, gewinnt Gestalt und beginnt zu sein.

Kapitel 7: Architektur

… in dem die Oberflächenbeschreibung als formales Entwicklungsartefakt in die Softwarearchitektur aufgenommen wird.

Kapitelziele

- *Verwalten der UI-Beschreibung zusammen mit dem Quellcode der Anwendung.*
- *Inhalte der UI-Beschreibung automatisiert in die Programmierung überleiten.*
- *Kommunizieren zwischen Projektbeteiligten über generierte User Interface-Prototypen.*

Technisches Regelwerk für Entwicklungs- und Laufzeitumgebung

Was das Projektmanagement für die Verwaltung und Organisation eines Softwareprojekts auf der Auftragsebene leistet, wird auf der Ebene der Entwicklungsorganisation und -technik von den Architekten geleistet.

Die Softwarearchitektur stellt den organisatorischen Rahmen und das technische Regelwerk für das Entwicklungsteam auf.

Software-Architektur

Die ***Softwarearchitektur*** teilt eine Software in Komponenten auf. Sie bestimmt den Zuschnitt, die Aufgabenteilung und die Zusammenwirkung der Softwarekomponenten in der Laufzeitumgebung. Zur Softwarearchitektur gehört auch die Organisation und Verwaltung der Quellartefakte in der Entwicklungsumgebung.

Informations-Architektur

Die ***Informationsarchitektur*** bezeichnet die Kombination aus Strukturierung von Wissen, Organisation von Daten und Benutzerinteraktionen zum Zugriff auf diese Informationen. Zu den Aufgaben eines Informationsarchitekten gehört der Aufbau von Informationsmodellen sowie von Repräsentationsformen für das Auffinden und Verwenden (z.B. gruppieren, navigieren, verknüpfen, indizieren) von den im Informationsmodell gespeicherten Informationen.

7.1 Integration in die Entwicklungsumgebung

Die Architektur ist eine große Klammer um die Bestandteile der Anwendung, die Entwicklungswerkzeuge und die Entwicklungstechnik. Was innerhalb der Architektur berücksichtigt und geregelt ist, gilt als architekturkonform und damit als technisch legal.

Alles andere wird als Wildwuchs von der technischen Leitung des Entwicklungsteams mehr oder minder rigoros abgelehnt. Im

Verhandlungsritual um bislang wenig bekannte Features und um neue Anwenderforderungen sind Hinweise auf technische Machbarkeit und auf die Architekturkonformität stets im Repertoire der Entwicklerargumente. Sie haben den Zweck, das technische Regelwerk stabil und konsistent zu halten.

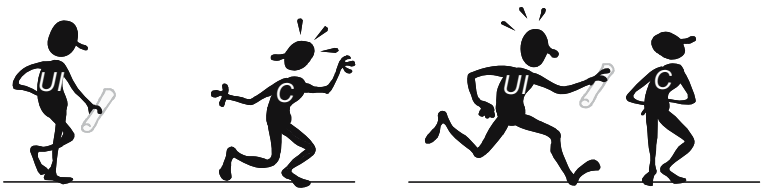


Bild 71: Fehlende Integration der UI-Modelle in der Architektur

User Interface-Anforderungen sind eine primäre Quelle von architekturrelevanten Änderungen an Daten, Funktionen und Abläufen der Software. Eine UI-Beschreibung enthält notwendigerweise Informationen, die die Funktionsmodelle und damit die Architektur und die Codierung unterschiedlich stark beeinflussen. Zudem ändert sich die UI-Beschreibung im Lauf ihrer Detaillierung.

Es liegt daher im Interesse der kodierenden Techniker, die UI-Beschreibung möglichst änderungsrobust in die Architektur zu integrieren.

Es ist zum Beispiel zweckmäßig, die UI-Modelle genauso wie andere Quellartefakte im Versionskontrollsystem (VCS) der Entwicklung zu verwalten.

Es ist ebenfalls im Interesse der UI-Entwickler (Anwendervertreter, UI-Spezialisten, Spezifizierer), dass die UI-Beschreibung im VCS der Entwickler beheimatet wird. Sie wird dadurch symbolisch und praktisch als entwicklungseigenes Artefakt und als Bestandteil der Architektur angenommen.

Sobald die Grenze von einem zugelieferten Anforderungsdokument zu einem Bestandteil der Systemmodelle mental überschritten ist, stehen psychologisch wie technisch viele nutzbringende Unterstützungsmöglichkeiten für die Zusammenarbeit der UI-Spezialisten und der Programmierer bei der Verwertung der UI-relevanten Informationen offen.

Bei vollzogener Integration der UI-Modelle in der Architektur laufen UI-Entwicklung und UI- Codierung Hand in Hand.

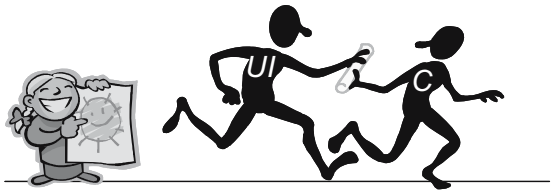


Bild 72: UI-Spezifikation und UI-Codierung laufen Hand in Hand.

Wenn die UI-Spezifikation ein Bestandteil der Softwarearchitektur ist, kann man folgende Vorteile nutzen:

- Die UI-Beschreibung wird im gesamten Entwicklungsteam als relevantes Teilmodell des Systems wahrgenommen
- Gemeinsame Versionskontrolle und Änderungsmanagement für alle Modelle
- Das Entwicklungsteam kann laufend - teilweise automatisiert - über Änderungen an der UI-Beschreibung informiert werden
- Laufende Abstimmung der Modelle aufeinander begünstigt das gemeinsame Verständnis der Anwendungsfunktionen und ihrer Nutzung an der Oberfläche
- Aus der UI-Beschreibung können jederzeit - teilweise automatisiert - Informationen und Steuerungsdaten für die Programmierung extrahiert werden

7.2 Das MVC-Entwurfsmuster

Modell-View-Controller-Architektur

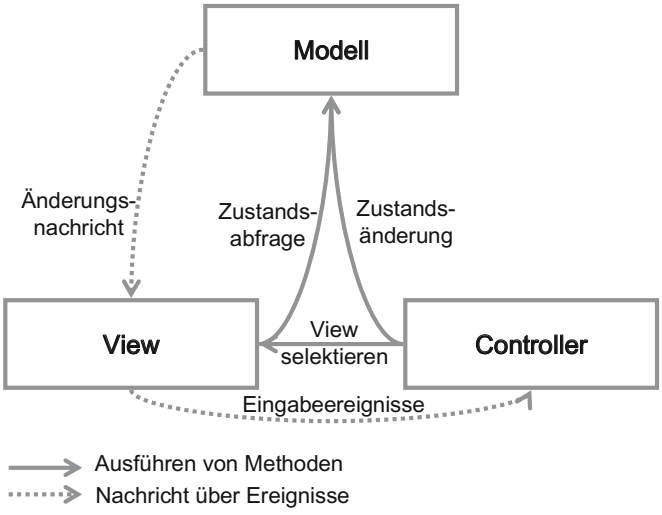


Bild 73: MVC-Entwurfsmuster

Die obige Grafik illustriert das Prinzip des MVC-Entwurfsmusters.

Der ***View***

- Präsentiert Modellinhalte
- Fordert vom Modell Aktualisierungen an
- Sendet Eingabeereignisse zum Controller
- Ermöglicht das Selektieren eines Views

Das ***Modell***

- Kapselt anwendungsinterne Zustände
- Beantwortet Zustandsanfragen
- Stellt die Funktionalität der Anwendung bereit
- Benachrichtigt den View über Änderungen

Der ***Controller***

- Definiert das Verhalten der Anwendung
- Ordnet Anwenderaktionen zu Modellaktualisierungen zu
- Wählt den View zur Beantwortung der Eingabeereignisse aus

Ziele des Einsatzes von MVC sind:

- Trennung von Anzeige, Präsentations- und Geschäftslogik sowie
  Datenzugriff
- Mandantenfähige Anwendung
- Wiederverwendbare Actions

Das MVC-Entwurfsmuster wurde mit der Verbreitung von Small-Talk bekannt. Es unterscheidet zwischen

- View: Präsentation (Ausgabe, Layout)
- Controller: Programmsteuerung (Interpretieren von Benutzeraktionen) und
- Modell: Objektmodell (Anwendungsfunktionalität, Datenhaushalt).

Document View

Document View ist eine von Microsoft in MFC (Microsoft Foundation Classes) umgesetzte Variante von MVC.

Eine weitere Variante des MVC-Entwurfsmusters ist das MVC2-Modell. Es ist eine für Webanwendungen optimierte zustandslose Variante des objektorientierten MVC-Entwurfsmusters.

In [Ceri 2003] wird im Detail beschrieben, wie die Modell-View-Controller-Architektur auf Webanwendungen adaptiert wird und

wie das Entwurfsmuster mit einer Modellierungssprache für datenintensive Web-Applikationen verknüpft werden kann.

MVC in der UI-orientierten Architektur

Im Zusammenhang mit der User Interface-Architektur ist das Verknüpfen des MVC-Entwurfsmusters mit User Interface-Modellen von Interesse für die automatisierte Code-Erzeugung. Wenn möglich, soll aus einem UI-Modell der Code für View und Controller sowie die Mapping-Schicht zwischen Controller und Model (funktionale Schicht mit Datenhaushalt und Business-Logik) generiert werden können.

Das MVC-Entwurfmuster unterteilt die Anwendung in drei Aufgabenbereiche:

- Modell (die eigentliche Funktionalität und Datenhaushalt),
- View (die Präsentation der Daten an der Oberfläche) und
- Controller (die Vermittlungsschicht zwischen View und Modell).

Die gängigen Methoden und Vorgehensweisen der Systemmodellierung (z.B. UML) detaillieren den ***Modell-Anteil*** der Anwendung.

UI-Modelle detaillieren den ***View-*** und den ***Controller-Anteil*** der Anwendung als Verbund.

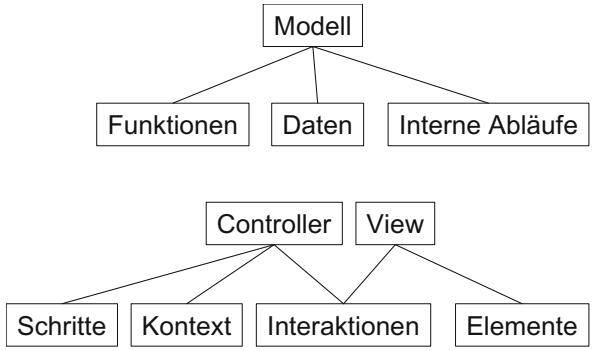


Bild 74: MVC und Zuordnung von Entwicklungs-Themen

Bei nicht UI-orientierten Architekturen sind Controller die aufwändigsten Teile der Anwendung und besonders änderungsanfällig. Selbst kleinere Änderungen an der Oberfläche, z.B. das Verschieben eines Eingabefeldes von einem Bildschirm in einen anderen, kann zu aufwändigen Änderungen in der Controllerschicht führen (siehe auch [Lauer 2002, S.29f]).

Die Benutzerschnittstelle ist der Teil der Anwendung, der Funktionen und Daten verwendet. Um die Verwendung zu steuern,

werden an der Oberfläche Informationen über den Kontext (zum Beispiel: Welche Funktionen stehen zur Verfügung? Wo im Bearbeitungsfluss befindet sich der Anwender? Welche Daten liegen bereits vor, welche fehlen noch?) benötigt.

Dieser Blickwinkel ermöglicht ein differenzierteres Bild der Aufgaben eines Controllers. Die Aufteilung der Controllerfunktionen in Schrittesteuerung, Kontextermittlung und Interaktionshandling ermöglicht robustere Controller und führt zu einem ergänzten Komponentenbild.

Kontexthandler

Beispielsweise kann die Kontextermittlung als ein zentraler Dienst angeboten werden, statt sie über die Controller zu verteilen. Oft werden in verschiedenen Controllern ähnliche Kontextinformationen ermittelt, z.B. ob die Mussfelder des Kundendatensatzes bereits erfasst und noch gültig sind. Da die Information nur einmal pro Anwendersitzung aus persistierten Daten ermittelt werden muss, bietet sich hierfür eine Kontextvariable an, die in einem zentralen Kontexthandler zur Verfügung gestellt werden kann.

Bauen verschiedene Kontextinformationen aufeinander auf, dann können sie bereits vorliegende Kontextvariablen heranziehen und ermöglichen eine wesentlich änderungsrobustere Controllerlogik. Gleiches gilt für die Übermittlung von eingegebenen Daten an die Anwendung. Die Prüfung der Vollständigkeit und Plausibilität der Eingaben sowie die Weitergabe an die Persistenzschicht können ebenfalls im zentralen Dienst eines Kontexthandlers untergebracht werden. Damit können Eingabefelder beliebig auf eine andere Dialogseite verschoben werden, ohne dass die Dialoglogik angepasst werden muss.

Die Anwendung der UI-orientierten Architektur auf das MVC-Entwurfsmuster ermöglicht Produktivitätssteigerung durch:

- Direkte Ableitung der Präsentationsschicht aus den Anforderungen (=> View)
- Erzeugung der Validierungs- und Mappingmechanismen (=> Controller)
- Erzeugung des Gerüsts für ein Klassenmodell (Daten und Funktionen) (=> Modell)

UI-Anteil in Dialogintensiven Anwendungen

Dialogintensive Geschäftsanwendungen (z.B. Verwaltungssoftware, Kreditsachbearbeitungssoftware) bestehen in der Regel zu mehr als 3/4 aus Dialogmasken und Mapping. Bei solchen Anwendungen entfallen weniger als 1/4 des Gesamtumfangs auf die Geschäftslogik. Die UML bietet hierfür keine Hilfestellung an, denn Oberflächenabläufe, Anwendungsnavigation und Präsentationsinhalte können in UML nicht direkt modelliert werden (siehe auch Kapitel „Sprache“).

Gerade aber eine Modellierungsmöglichkeit der Präsentationsschicht (d.h. der Mensch-Maschine-Schnittstelle, also des „View“ und des „Controllers“ aus dem MVC-Entwurfsmuster) ist aus Sicht einer dialogintensiven Geschäftsanwendung besonders wünschenswert und führt zur Steigerung der Produktivität.

7.3 Model Driven Architecture (MDA)

Heute (2006) ist es seit mehr als 25 Jahren üblich, Anwendungen mit problem- und objektorientierten Sprachen (Cobol, Pascal , C, C++, Java) zu programmieren. Die i.d.R. objektorientierten Programmiersprachen sind mit Bibliotheken ausgestattet, die [unter anderem] Datenzugriff und [grafische] Oberflächen IO unterstützen.

Moderne Entwicklungswerkzeuge verwenden zum Teil schon Modelle, zum Beispiel UML-Klassendiagramme, um daraus die Datenhaltungs- und die Datenzugriffsschicht als Code zu generieren.

Üblich sind auch damit einhergehende UI-Designwerkzeuge zur visuellen Entwicklung von grafischen Dialogseiten. Gewöhnlich wählt man Oberflächenelemente aus einem vordefinierten erweiterbaren Vorrat aus, platziert sie mit drag und drop auf der Dialogseite und legt in ihren Properties fest, wie sie aussehen, welche Daten sie anzeigen und welche Funktionen sie auslösen sollen. Aus diesen Masken-Modellen wird ebenfalls 1:1 Code generiert.

Umsetzung von MDA-Paradigmen in der Praxis

Zur heutigen Landschaft der Anwendungsentwicklung gehören auch die Paradigmen der MDA, welche uns nahelegt, weite Teile der Anwendung in Form von oben erwähnten und weiteren Modellen zu beschreiben und daraus den herkömmlichen Programmcode und/oder Steuerungsdaten zu generieren.

Als Kriterien für das Anwenden der MDA werden dabei

- Kapselung von Implementierungsdetails (also Komplexitätsreduktion und Wiederverwendbarkeit),
- Redundanzvermeidung (also Integritäts- und Qualitätserhöhung) sowie die
- Generativität (also Robustheitssteigerung und Senkung von Änderungskosten).

Das klingt gut: Höhere Produktivität in höherer Qualität zu kleineren Kosten. Fast wie New Economy.

Nur die Frage danach, was wir modellieren sollen und was woraus sollen wir was generieren, beantwortet uns die MDA-Evangelisten mit Schulterzucken - na eben beliebig - ois modellieren - ois generieren.

Kommen wir aber aus dem Kontext der heutigen Landschaft der Softwareentwicklung zurück zu den Anforderungen an MDA-freundliche Werkzeuge zur UI-Entwicklung.

Eine Form der MDA ist, dass man Codeschnipsel, die sich auf der Implementierungsebene wiederholen, ins Modell schiebt, und damit die Implementierung parametrisiert. Hierbei entstehen Konfigurationsdateien.

Eine andere sind ***DSLs*** (Domain Specific Languages - fachgebietspezifische Sprachen). Hier beschreibt man mit einem ***Fachvokabular*** fachliche Sachverhalte (zum Beispiel: Periodensystem der Elemente, technisches Zeichnen, Elektrobaupläne) und bildet diese Information mit Generatoren in Code der Anwendung ab (z.B. Funktionen zur Ausgabe der Periodentabelle, der Kernschalengrafiken sowie Fusion und Spaltung von Elementen).

Hierbei entstehen fachspezifische Modelle von Fachdaten und Fachalgorithmen oberhalb der Ebene der Programmiersprache.

Die Model Driven Architecture ist noch auf dem Weg von einer Initiative zu einem zuverlässigen Standard. Heute zeigt die Arbeit mit MDA Parallelen zum frühen Stadium des Dieselmotors, wenngleich dies wahrscheinlich nicht mit einer eigenen Gedenktafel prämiert wird: Im MAN Diesel-Werk in Augsburg hängt an einer Ziegelmauer eine in Messing gefasste Gedenktafel, welche besagt, dass an dieser Stelle Herr Ingenieur Diesel seinen ersten Motor in Betrieb nahm, welcher 30 Sekunden lief, bevor er explodierte. Heute aber ist der Dieselmotor technisch ausgereift und funktioniert zuverlässig in unterschiedlichsten Fahrzeugen.

Für die MDA in Verbindung mit ihrem Treibstoff - den domänenspezifischen Sprachen - ist dieser Reifeprozess aus meiner Sicht noch im Gange.

7.4 UI-orientierte Laufzeitkomponenten

Präsentationsobjekte
UI-Objekte
Geschäftsobjekte
Systemobjekte

In einer User Interface-orientierten Architektur kann die Oberfläche Komponenten verwenden, die Dienste zum Austausch von Informationen mit dem funktionalen Anteil der Anwendung zur Verfügung stellen.

Diese Komponenten könnten heißen:

- ***Funktionsmapper*** (bildet Requests zum Ausführen von Funktionen auf parametrisierte Methoden der Anwendung ab)
- ***Valuemapper*** (bildet Requests zum lesenden oder schreibenden Zugriff auf Daten auf das Objektmodell der Anwendung ab)

- ***Listenmapper*** (bildet den Zugriff auf Listen und Listeneinträge auf Listenobjekte ab)
- ***Situationsmapper*** (bildet Requests zum Abfragen von Zuständen auf Wahrheitswerte und logische Ausdrücke ab)

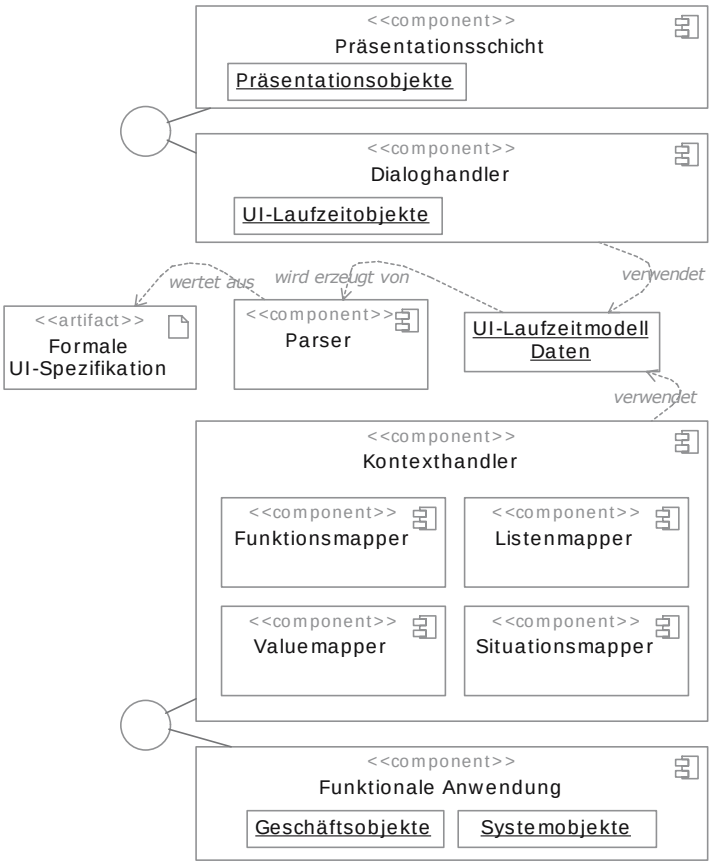


Bild 75: Komponenten-Modell für eine UI-orientierte Architektur

Die UI-orientierten Laufzeitkomponenten können in einem so genannten ***Kontexthandler*** zusammengefasst werden. Der Kontexthandler bewältigt die Aufgabe, die Oberfläche mit Informationen über den Zustand der anwendungsinternen Geschäfts- und Systemobjekten zu beliefern.

In umgekehrter Richtung löst der Kontexthandler die Anforderungen von Funktionen seitens der Oberfläche in das Ausführen und das Parametrisieren einzelnen Methoden auf.

7.5 Ausleiten von Prototypen

Aus den in der UI-Beschreibung enthaltenen Informationen kann man Bildschirme und einfache Kontrollanweisungen herausparsen und daraus UI-Prototypen erzeugen.

Ein zum Beispiel in HTML und JavaScript erzeugter Prototyp eignet sich gut als Kommunikationsplattform für die an der Konstruktion eines User Interface beteiligten Personen.

Mit CSS (Cascading Style Sheets) kann das Erscheinungsbild des Prototypen jederzeit bequem und meist ohne erneute Generierung des Prototypen verändert werden. Der Kunde oder Begutachter des UI-Entwurfs braucht keine besondere Laufzeitumgebung, um den Prototypen auszuführen, also kann man den Prototypen z.B. einfach per Mail an den Gesprächspartner verschicken.

7.6 Weitere Transformationen

Man kann aus einer UI-Beschreibung noch weitere für Entwickler nützliche Dokumente und Aufstellungen ausleiten.

- Data Dictionary
- Funktionslisten
- Elementlisten
- Wertebereiche

Prinzipiell kann eine eindeutige UI-Beschreibung in einen beliebigen Datenstrom oder in einen beliebigen Code umgewandelt werden. Sofern die Semantik (die Bedeutung) der Bestandteile einer UI-Beschreibung eindeutig ist, können daraus beliebige andere Artefakte generiert werden.

Über Dinge, die in das Zielmodell hineingeneriert werden (z.B. Seitenlayout, Rendering einzelner Bedienungselemente, Spezifikationsauslassungen), aber im Quellmodell nicht vorhanden sind, müssen Annahmen getroffen werden.

Die Annahmen können innerhalb der Transformationsprozedur (also im Code) oder in einer Parameterdatei (z.B. eine XML-Konfiguration für den Transformator) abgelegt werden.

7.7 Nutzen

Der wesentliche Vorteil der Einbindung einer (formalen) UI-Beschreibung in die Architektur der Software liegt darin, dass

Informationen über Änderungen an der UI-Spezifikation unmittelbar für die Programmierung zur Verfügung stehen.

Diese Informationsbereitstellung kann bei einer formalen UI-Spezifikation zu einem beliebigen Grad automatisiert werden. Es können die in der UI-Beschreibung enthaltenen Regeln als XML-Daten bereitgestellt oder von einem Parser zur Codeerzeugung verwendet werden. Aus der UI-Beschreibung können bereits in frühen Projektphasen Oberflächenprototypen erzeugt und zur Abstimmung und Absicherung der geplanten Funktionen verwendet werden.

Der nichttechnische, jedoch nicht minder bedeutsame Vorteil der Beheimatung der UI-Modelle in der Softwarearchitektur ist, dass damit die ganze Anwendung von der Architektur umfasst wird. Damit wird auch die psychologische Lücke zwischen Oberfläche und Funktionalität geschlossen.

Die Oberfläche ist kein Aufsatz auf die Funktionalität, sondern die Schnittstelle zu ihrer Nutzung. Benutzerschnittstelle und Funktionalität können optimal nur in einem gemeinsamen - von der Architektur getragenen - Kontext entwickelt werden.

Kapitel 8: Funktionsmodellierung

… in dem UI-Modelle und UML zusammen angewandt werden, sich ergänzen und gemeinsam wirken.

Kapitelziele

- *Kooperative Modellierung von Benutzeroberflächen und von funktionalen Anwendungsschichten.*
- *Mapping von Oberflächeneigenschaften auf Eigenschaften der funktionalen Anwendung (z.B. auf Modellebene: Welche Funktionen werden in der Oberfläche vorausgesetzt und welche werden von der Anwendung bereitgestellt?)*
- *Übernahme der UI-Beschreibung als Gerüst für funktionale Modelle (z.B. Klassenliste oder Data Dictionary).*

Der Begriff ***Funktionsmodellierung*** bezeichnet im Folgenden das Modellieren von internen Abläufen und Strukturen der funktionalen Anwendung. Damit ist auch die Modellierung von Daten bzw. Objekten gemeint. Die ***funktionale Anwendung*** ist der Teil der Software, der unter der Oberfläche liegt. In betriebswirtschaftlichen Anwendungen wird diese Schicht der Software oft als Geschäftslogik oder Fachfunktionalität bezeichnet. In technisch orientierten Anwendungen wird die funktionale Anwendung meist als Systemlogik bezeichnet.

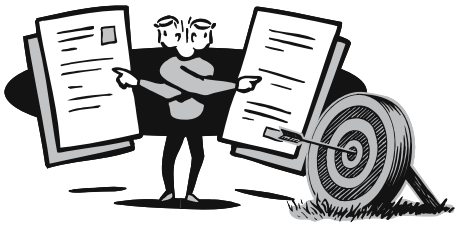


Bild 76: Oberfläche, Funktionen und Daten sind Teile eines Ganzen.

Oberfläche und funktionale Anwendung bzw. das Modellieren der Softwareoberfläche und die Funktionsmodellierung haben verschiedene Berührungspunkte.

- An der Oberfläche werden Funktionen und Daten des Softwaresystems verwendet.
- Beim Beschreiben einer Oberfläche setzen die Spezifizierer die Existenz der verwendeten Daten und Funktionen voraus und treffen Annahmen über Datenstrukturen und Funktionsweisen der Anwendung.

- Beim Modellieren der Oberfläche schließt man also von der Bedienung auf Funktionen und Daten bzw. auf Objekte.
- Beim objektorientierten Modellieren ist es umgekehrt: Man schließt von den Funktionen und Datenstrukturen auf ihre Verwendung. Daher ergänzen sich beide Ansätze.

Entwickler treffen Annahmen

UI-Entwickler treffen Annahmen über Daten und Funktionen der Anwendung. ***Objektentwickler*** treffen Annahmen über Verwendung der Objektinstanzen an der Oberfläche.

Es liegt nahe, die Annahmen aus der User Interface-Modellierung mit Klassenmodellen, Use Case-Diagrammen, Aktivitäts- und Sequenzdiagrammen abzusichern. Die Modelle der Oberfläche und die Modelle der Funktionen und der Daten ergänzen und überprüfen sich gegenseitig.

8.1 Abhängigkeiten zwischen Bedienung und Funktion

Bedienung und Funktion einer Software bedingen sich gegenseitig. Oft kann das Was und das Wie einer Software nur im Zusammenhang verstanden werden, daher ist es auch ratsam, UI-Modell und Funktionsmodelle in gegenseitiger Abstimmung zu entwerfen.

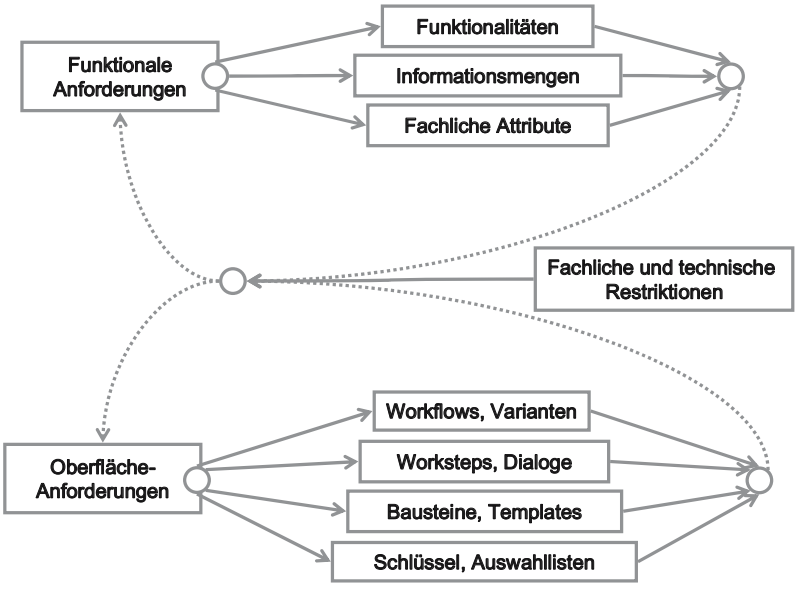


Bild 77: Gegenseitige Abhängigkeiten von Bedienung und Funktionen

Oberflächenanforderungen haben mittelbaren Einfluss auf funktionale Anforderungen. Umgekehrt wirken sich Funktionen auch auf die Art der Bedienung aus. Die fachlichen und technischen Restriktionen wirken sich sowohl auf die Bedienung als auch auf die Funktionen aus.

User Interface und Funktion bedingen sich gegenseitig

Inhaltliche Anforderungen stehen in Wechselwirkung mit Nutzungsanforderungen, d.h. sie bedingen sich zum Teil gegenseitig.

Das in der herkömmlichen Vorgehensweise postulierte, voneinander getrennte Betrachten dieser beiden Bereiche ist nicht praktikabel.

Mit der UI-Orientierung wird dem Vorliegen dieser Wechselwirkungen Rechnung getragen und entsprechende Behandlungsmöglichkeiten für die sich daraus ergebenden Aufgaben aufgezeigt.

8.2 Kollaboratives Modellieren

Durchgängigkeit

Eine durchgängige Spezifikation erstreckt sich von der Präsentationsschicht bis zur Systemschicht der Anwendung. Durchgängig bedeutet, dass alle Aspekte der Software in einem gemeinsamen konzeptionellen Rahmen gehalten festgelegt werden und zueinander in Beziehung stehen.

Der Vorteil der Durchgängigkeit ist, dass das Modell der Benutzeroberfläche und die Modelle des inneren Aufbaus des Systems aufeinander abgestimmt sind und jeweils anhand des anderen Modells auf Vollständigkeit und Korrektheit überprüft werden können.

In einem solchen Modellverbund ist es weitgehend sichergestellt, dass keine Bedienungselemente, Ablaufschritte, Funktionen und Daten vergessen wurden. Ein weiterer Vorteil der Durchgängigkeit liegt darin, dass so genannte tote Enden, also Daten und Funktionen, die nicht verwendet werden, oder aber Bedienungselemente, die nicht mit Funktionen unterlegt sind, so vermieden werden können.

UI-Modell und Modelle des Systeminneren müssen aufeinander abgestimmt sein.

Das kollaborative Modellieren kann man vereinfacht auf die folgende Formel bringen: Anwendungsabläufe und funktionelle Abläufe müssen wie Schraube und Mutter zueinander passen. ***UI-Modell*** und ***Modelle des Systeminneren*** werden voneinander getrennt erstellt, müssen aber fortwährend aufeinander abgestimmt werden.

Eine Aufgabenteilung kann beim kooperativen Modellieren wie folgt aussehen:

UI-Modell

- Schwerpunkt: Anwendungsabläufe und Präsentationsinhalte
- Vorgehen: Workflow- und anwenderorientierte Modellierung, z.B. mit Maskenentwürfen und Lucia
- Ergebnis: Oberfläche und Präsentationslogik

Modelle des Systeminneren

- Schwerpunkt: Funktionelle Abläufe, Inhalte und technische Umsetzung
- Vorgehen: Objekt- und komponentenorientierte Modellierung, Modellierung der systeminternen Abläufe, in der Regel mit UML.
- Ergebnis: Geschäftslogik, Kontexthandling, technische Komponenten

Gemeinsam liefern die Modelle die für eine händische oder eine modellgetriebene Implementierung notwendigen Informationen.

Beispiel: Login Prozedur

Eine durchgängige Spezifikation für eine Login-Prozedur: Oberflächenbeschreibung, Ablauf, Funktionen und Daten.

Strukturperspektive

```
Start Step [Anwendungsrahmen]
  {
  on Enter perform Step ~[LoginProzedur] !! (Zuerst einloggen)
  on Signal !! (nach dem Einlogversuch prüfen ob eingeloggt)
    when ~[not situations.eingeloggt]
    exitStep
  }

Service Step [LoginProzedur]
  {
  Step [Login-Eingabe] !! (Anwender identifiziert sich)
  Service Step [Login-In-Progress] !! (Fortschrittanzeige)
  Service Step [Login-Error] !! (Fehlermeldung im Misserfolgfall)
  }
```

Dialogperspektive

```
Screen [Login-Eingabe]
  {
  !! (Eingabe- und Bedienelemente)
  Textinput [Login-Id]
  Textinput [Login-Password]
  Button [ok]
    on Command
      invokeFunction doLogin
      goto Step [~login-in-progress]
```

```
    on Situation ~[Login-Parameter-Eingegeben]
      inputmode =enabled
      else =disabled
  Button [abbrechen]
    on Command exit Step


  Situation [Login-Parameter-Eingegeben]=                    Lokaler Kontext
    ~[not Login-Id.empty] and ~[not Login-Password.empty]
  }

Screen [Login-in-Progress]
  {
  on Signal [if ~login-done]
    end step
  Text [progress-message] value=[~Login-Objekt.message-text]
  Progressbar value=[~Login-Objekt.progress-percentage]
  }

screen login-error
  {
  Display error-message value=[~Login-Objekt.message-text]
  Button [ok]
    on command exit Step
  }


Situation Eingeloggt=                                        Kontextperspektive
  ApplicationValue.LoginConfirmed= [ok]


Application-Interface Login-Objekt
  {
  method doLogin (id, password)
    use Login-Eingabe.Login-Id, Login-Eingabe.Login-Password
  string id
  string password
  string message-text
  string progress-percentage
  picture message.picture
  }
```

Aus dem Application-Interface und aus der globalen Situation „Eingeloggt“ lässt sich 1:1 eine Klassenbeschreibung formen.

8.3 Oberflächeneigenschaften mit UML-Modellen absichern

Die Anforderungen an die Bedienung können mit Klassenmodellen, Aktivitäts- und Sequenzdiagrammen auf Machbarkeit und Plausibilität überprüft werden.

Die in einem UML-Klassenmodell beschriebenen Strukturen und Beziehungen beinhalten implizite Regeln für den Zugriff auf

Objektinstanzen, z.B. wie man die Bestellungen eines bestimmten Kunden im Zeitraum X findet.

Die Objektstruktur legt damit der Anwendungsoberfläche Restriktionen auf. Eine Restriktion wäre zum Beispiel, inwiefern die Datenorganisation eine Kundensuche vor der Suche nach den Bestellungen dieser Kunden voraussetzt (Suche über Liste der Kunden-Id's) oder ob es reicht, das Produkt und den Zeitraum als Suchkriterien anzugeben, um alle Kunden, die dieses Produkt bestellt haben, zu finden.

Wenn die Datenstruktur nicht entsprechend optimiert ist, kann sich die Identifikation von Kunden anhand von bestellten Produkten als ineffizient erweisen, und die Oberfläche sollte anders aufgebaut oder die Datenorganisation modifiziert werden.

Die Entscheidung, was geändert wird, muss von den Verantwortlichen für UI und Datenorganisation gemeinsam und unter Abwägung des Aufwands und der Arbeitsweise des Anwenders getroffen werden.

8.4 Funktions- und Datenmodelle ableiten

Beim Modellieren des Kontextes der Oberfläche wird die Erwartungshaltung an das Vorliegen von Daten und Funktionen beschrieben.

Aus diesen Informationen kann ein Gerüst für das UML-Klassenmodell erzeugt und anschließend in einem UML-Werkzeug weiter detailliert werden.

Beispiel: Ableiten des Gerüsts für ein Klassenmodell aus dem UI-Modell der Kundenverwaltung.

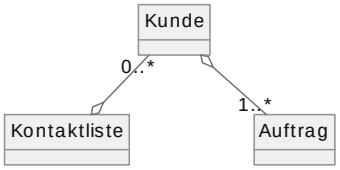


Bild 78: Klassengerüst für die Kundenverwaltung

Die Beziehung zwischen UI-Informationen und anwendungsinternen Informationen kann mit Hilfe von Mapping-Tabellen abgebildet werden.

Beispiele für Mapping-Tabellen:

- Datenverwendung ←→ Datenstrukturen
- Datenverwendung ←→ Datenselektion-Statements

- Anzeigeausschnitt ←→ Datenvorrat
- Funktionsverwendung ←→ Methoden
- Funktionskontext ←→ Methodenparameter

8.5 Nutzen

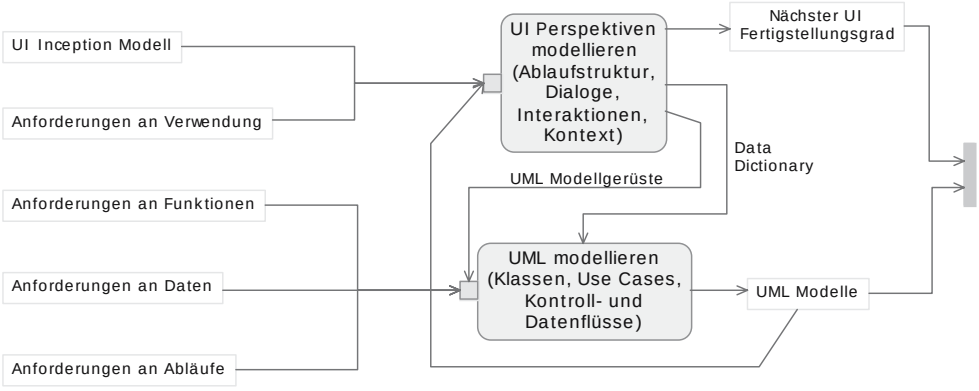
UI-Modellierung verträgt sich gut mit den allgemein verwendeten Methoden (UML, RUP, V-Modell) und ergänzt diese um UI-orientierte Methoden und Vorgehensweisen. UI-Modellierung fördert die Kommunikation im Projekt und unterstützt die Erstellung der funktionalen Modelle (z.B. durch die Verwendungssicht der geforderten Daten und Funktionen).

Durch Verbindung von UML-Modellen mit UI-Modellen erhält man ein in sich geschlossenes, abgesichertes und rundes Gesamtbild der Anwendung.

Ich empfehle Ihnen, Oberfläche und Anwendungsinternas parallel zu entwerfen und kontinuierlich miteinander abzugleichen. Sie erreichen so ein konsistentes und fehlerarmes Ganzes.

Die Abbildung zeigt eine Vorgehenskarte für das Verbinden der UI-Modellierung mit der Funktionsmodellierung.

Bild 79: Vorgehenskarte für synchrone UI- und UML-Modellierung



Verzichten Sie auf die Forderung der klassischen Anforderungsanalyse, dass nur das „Was“, nicht aber das „Wie“ der Anwendung erhoben werden soll.

Bloss nicht auf die Lösung eingehen?

Die sicher berechtigte Sorge, nicht in technischen Umsetzungsdisputen zu versinken, wird von Scharbert hervorgehoben: *„Da der Analytiker zur **Lösungsneutralität** verpflichtet ist, befasst er*

sich nicht mit technischen Repräsentationen; zumindest gibt er es nicht laut zu.“ [Scharbert 2005, S.129].

Beliebte Standardfrage: „Na, sieht man schon was?“

Andererseits: Welcher Entwickler hat noch keinen Teamleiter erlebt, welcher, plötzlich hinter einem stehend, mehrmals täglich die wohlbekannte Frage „***Sieht man schon was?***“ stellt? Jeder am Projekt Beteiligte will das Produkt anfassen und erleben können, wieso also dem Analytiker das Ganzkörperpräservativ der Lösungsneutralität um jeden Preis überziehen?

Aus meiner Sicht ist es sachdienlich, das Was vom Wie zu trennen, aber nicht zeitlich, sondern nur räumlich, das heißt: Die Analyse klar nach „Was“ und „Wie“ gliedern.

Auf Merkmale der Lösung eingeben!

Zum Beispiel: Wenn Sie einen Tisch bauen sollen, wird es Ihrem Kunden und Ihnen schwer fallen, nicht darüber zu diskutieren, wie der Tisch verwendet wird, wenn er fertig ist. Der Kunde und der Entwickler des Tisches werden darüber sprechen, wo der Tisch stehen wird, was auf ihm stehen wird und wer daran sitzen wird. Dabei werden Sie auch über das Holz und das Metall sprechen, aus dem der Tisch beschaffen sein könnte, und eventuell ihre Vorstellung durch Beispiele echter Tische untermalen.

Informationen über Verwendung mit funktionalen Anforderungen verknüpfen.

Verhindern Sie als Analytiker Beschreibungen und Diskussionen über das Verwenden von Oberflächeneigenschaften und über technische Umsetzungswünsche nicht, sondern verknüpfen Sie Verwendungsinformationen mit funktionalen Anforderungen. Im Beispiel des Tisches wäre z.B. die Anzahl der Beine, die Form, die Oberflächenbeschichtung, die Höhen- und Neigungsverstellbarkeit aus der Art der Nutzung des Möbels ableitbar.

Lassen Sie sich beim Analysieren und Dokumentieren der Anforderungen von diesem Leitspruch begleiten: *„Wenn du ein Schiff bauen willst, musst du die Leute nicht zum Baumfällen antreiben, sondern ihre Sehnsucht nach dem Meer wecken“ (Antoine de Saint-Exupéry).*

Durch paralleles Entwerfen von Verwendungsaspekten und der technischen Umsetzung erhalten Sie zwei Teilmodelle:

- Modell der Anwendungsoberfläche
  - Präsentationslogik,
  - Dialogseiten,
  - Oberflächenabläufe) und
- Modell des „Anwendungsinnenlebens“
  - Datenmodell,
  - Funktionsvorrat,
  - interne Abläufe,

die sich gegenseitig ergänzen und bestätigen.

Mit Hilfe der so verknüpften Modelle können Analytiker konkrete Eigenschaften der Anwendung verständlich vorstellen und mit den Projektbeteiligten diskutieren.

So kann das Entwicklungsteam Widersprüche zwischen Verwendungsabsicht und Konstruktionsentwurf aufspüren und noch vor der Implementierung korrigieren.

Die parallele Verwendung der User Interface-orientierten Modelle und der UML-Modelle führt zu besten Ergebnissen.

Die Mühe des kontinuierlichen Abgleichens zwischen den Teilmodellen wird durch kürzere Entwicklungszeit, weniger Nachbesserungen und höhere Qualität des Softwareprodukts entlohnt.

Kapitel 9: Prozesssteuerung

… in dem Informationen aus UI-Modellen zu elektronischen Workflows destilliert werden.

Kapitelziele

- *Visualisieren der Anwendungsabläufe durch Prozess- und Workflowgraphen.*
- *Modellieren von vorgegebenen Prozessen in der Anwendungsoberfläche.*
- *Verwerten der ablaufrelevanten Informationen als Steuerungsregeln für Ablaufmaschinen (z.B. externe Workflow-Systeme, anwesndungsinterne Ablaufsteuerung).*

In der UI-Beschreibung sind sämtliche Ablaufmöglichkeiten der Anwendung festgelegt. Sie sind darin nicht explizit (etwa als Transitionsgraph) ausformuliert, sondern in Form von Transitionen zwischen Dialogseiten enthalten. Prozesse und die Ablaufsteuerung von Anwendungen hängen immer enger zusammen.

Anwendungen sollen Geschäftsprozesse unterstützen

Unternehmen, die große Verwaltungsanwendungen betreiben, erheben oft die Forderung, dass die Anwendung den Prozessen des Unternehmens folgen muss und nicht umgekehrt. Meist wird diese Forderung mit einer Prozessbeschreibung (z.B. in einem Aris-Modell oder in IDEF) unterlegt.

Hersteller von Prozessmodellierungs- und BPM- (Business Process Management) Werkzeugen werben zuweilen mit dem Slogan: Eine Anwendungssoftware ist eine ausführbare Prozessbeschreibung. Auch wenn die Formulierung übertrieben erscheint: In Anwendungen, die etwa Verwaltungs-, Verkaufs- und Sachbearbeitungsvorgänge abbilden, gehört die Führung des Anwenders durch die einzelnen Schritte dieser Vorgänge - also eine Workflowsteuerung - auf jeden Fall dazu.

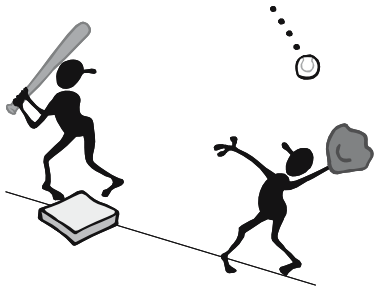


Bild 80: Abläufe und Regeln sind implizit im Programmcode eingebaut

Die Anwendung befolgt also eine Prozessbeschreibung. In den meisten Fällen ist diese Prozessbeschreibung implizit. Das heißt, dass die Abläufe und Regeln im Programmcode der Anwendung abgebildet sind. Die Regeln sind aber nicht explizit formuliert, sondern auf die Ebene der Programmiersprache herunter gebrochen. Dementsprechend ist meist Programmierarbeit angesagt, wenn eine vorhandene Anwendung an Änderungen der Abläufe des Unternehmens angepasst werden soll.

9.1 Prozesslogik und Situationen

Prozesslogik

Der hier eingeführte Begriff ***Prozesslogik*** bezeichnet die Regeln, die den Bearbeitungsfluss einer Softwareanwendung steuern.

Zur Prozesslogik gehören insbesondere Abfolgen von Dialogseiten sowie das Auswerten und Setzen des ***Bearbeitungsstatus***, auf den die ***Geschäfts***- und ***Präsentationslogik*** der Anwendung Bezug nehmen kann, um Entscheidungen über Aktionen der Software zu treffen. Die Entscheidungen können zum Beispiel das Aufrufen einer bestimmten Systemfunktion oder das Ausblenden und Einblenden von Kontrollelementen betreffen.

Bearbeitungsstatus

Der Begriff ***Bearbeitungsstatus*** steht für eine Ansammlung von benannten ***Statusmarken***, die das Vorliegen verschiedener ***Situationen*** in der Softwareanwendung repräsentieren und die jeweils den Wahrheitswert „wahr“ oder „falsch“ haben können.

Die Werte der Statusmarken werden durch die Auswertung eines logischen Ausdrucks (einer „boolean expression“) gesetzt. Der Wert einer Statusmarke kann aus dem Datenhaushalt der Anwendung unter Verwendung der Werte anderer Statusmarken ermittelt werden. Das heißt: Durch Wiederverwendung der Statusmarken können komplexe Situationen aus einfacheren Situationen zusammengesetzt werden.

Eine Situation kann sich aus anderen Situationen zusammensetzen

Bei Änderungen in der Prozesslogik brauchen in der Regel nur die Statusmarken geändert zu werden, die die Basissituationen repräsentieren. Die zusammengesetzten Statusmarken, die auf einfachere Situationen aufbauen, bleiben in der Regel stabil.

Durch den Aufbau einer Hierarchie von Statusmarken zur Repräsentation der verschiedensten Situationen, über die die Software an der Oberfläche „Bescheid wissen“ muss, bleiben die Regeln, die die Prozesslogik verwendet, flexibel und äderungsrobust.

Beispiel: Situationen festlegen

Die folgenden Statusmarken definieren zwei zusammengesetzte und zwei einfachere Situationen. Die Situationen `s.buchung_ok` und `s.bestellung_ok` nehmen Bezug auf andere, weniger abstrakte Situationen.

Für die Situationen `s.kundendaten_ok` und `s.artikel_vorraetig` nehmen wir in unserem Beispiel an, dass sie durch Prüfung von mehreren Datenfeldern im Datenhaushalt der Anwendung ermittelt werden. Um das Beispiel einfach zu halten, nehmen wir des Weiteren an, dass diese Situationen keine weiteren untergeordneten Situationen einschließen.

```
s.buchung_ok ::=
  s.bestellung_ok & s.bestaetigung_erfolgt
s.bestellung_ok ::=
  s.kundendaten_ok & s.artikel_vorraetig & s.freigabe_erfolgt
s.kundendaten_ok ::=
  (weitere Detaillierung)
s.artikel_vorraetig ::=
  (weitere Detaillierung)
```

Wenn sich nun etwas an den Prüfbedingungen für Kundendaten ändert, dann muss nur diese eine Regel `s.kundendaten_ok` angepasst werden. Alle anderen Regeln, bei denen die erfolgreich stattgefundene Prüfung der Kundendaten Teil der Regel ist, sind automatisch aktuell und brauchen nicht geändert zu werden.

Prozesslogik steuert den Bearbeitungsfluss

Mit der Prozesslogik wird also ein Netzwerk aus Regeln für die Steuerung von Übergängen an der Oberfläche modelliert. Das Regelwerk kann aber auch für den anwendungsinternen Kontrollfluss bereitgestellt und verwendet werden.

Der richtige Zuschnitt des Netzwerks aus Regeln ist eine Aufgabe, die in der Verantwortung des Workflow-Modellierers liegen sollte. Die Aufstellung des Regelwerks sollte sehr umsichtig erfolgen und das Ownership bei nur einer Person liegen, damit das Regelwerk konsistent, effizient und robust bleibt.

Separation of concerns

Die algorithmische Komplexität einer Prozesslogik ist genauso hoch wie beim Programmieren. Allerdings ist die Form transparenter und die Beschreibung robuster als programmierter Code. Der Vorteil ist, dass die Regeln (die „Constraints“ und die „Concerns“) separiert vorliegen.

Wie im obigen Beispiel gezeigt, können im Prinzip beliebige Situationen als Statusmarken abgebildet werden. Die Statusmarken können sehr komplexe Situationen, z.B. `verkaufsvorgang_abgeschlossen` oder einfache Zustände, z.B. `nachname_eingegeben` abbilden.

Neben den vom abzubildenden Prozess abhängigen, individuellen Statusmarken bieten sich einige Standard-Statusmarken an, die für jeden logischen Schritt gelten, an. Dies sind zum Beispiel:

Interne Zustände von logischen Schritten

```
s.not_startable
s.startable
s.started
s.done_ok
s.done_needs_corrections
s.done_but_failed
s.done_but_failed_finally
```

Damit lässt sich für jeden logischen Schritt der Anwendung der aktuelle Status festlegen und feststellen. Auf der Basis dieser elementaren Statusmarken können dann die für die Anwendung spezifischen Situationen durch logische Ausdrücke beschrieben werden.

Weitere Beispiele:

```
fahrzeug_faehrt = ~data.geschwindigkeit <> 0
vorgang_abgeschlossen =
  ~situations.geprueft and
  ~situation.genehmigt and
  ~situation.gebucht
```

9.2 Ablaufsteuerungsdaten extrahieren

Situationen können als Entscheidungs-Variablen für User Interface und für Geschäftslogik dienen.

Die in der Prozesslogik definierten Statusmarken sind in fast allen Fällen genauso auch für die Geschäftslogik relevant. Es bietet sich daher an, das für das UI-definierte Regelwerk zur Workflowsteuerung auch für die Geschäftslogik bereitzustellen.

Der Nutzen ist hoch: Ein gemeinsames Regelwerk für UI und für interne Kontrollfluss-Entscheidungen senkt den Aufwand für Erstellung und Pflege der Anwendung. Die redundanzfrei definierten Regeln verhindern Inkonsistenzen zwischen UI und Geschäftslogik und sorgen für eine flexible und änderungsrobuste Anwendung.

Hierzu müssen die Prozessregeln und die Situations-Statusmarken aus der UI-Beschreibung herausgeparst und als Steuerungsdaten oder als generierter Code in der Entwicklungsumgebung der Anwendung bereitgestellt werden. Dieser Aufwand zahlt sich aus, weil das Prozessregelwerk nun beinahe beliebige Veränderungen an der Ablaufsteuerung ohne Änderungen am Programmcode ermöglicht.

Denkbar ist, dass der Kunde das Prozesslogik-Regelwerk selbst als Bestandteil der Anwendung pflegen und so Aufwände für zukünftige Anpassung sparen möchte.

Allerdings rate ich davon ab, das durchaus komplexe Regelwerk in der Fachabteilung zu pflegen oder die Regeln mal eben kurz im Produktivsystem zu ändern. Zirkuläre Regeln können zum Stillstand der Anwendung und zu anderen sehr unerwünschten Resultaten führen.

Das Werkzeug der Prozesslogik ist ein mächtiges Instrument zur Steuerung der Bearbeitungsflüsse im UI und in der funktionalen Schicht. Es ist aber auch mächtig komplex und sollte daher in der Entwicklungsumgebung wie Quellcode behandelt werden. Das Formulieren von Prozesslogik liegt innerhalb eines UI-Modells nahe am Programmieren, weil sich aus der Prozesslogik im Prinzip eine Sequenz von Anweisungen der Form `if (expression) then (reactions)` ergibt.

Die Bereitstellung der Prozesslogik für die Verwendung in der Entwicklungsumgebung kann in einem beliebigen Format erfolgen.

Die in der UI-Beschreibung aufgestellten Workflow-Steuerungsregeln lassen sich z.B. in eine Transitionstabelle der folgenden Form umformen:

Tabelle 1

step_id	signal_type	situation_idref	transition_list	parentstep_idref	idolstep_idref

Die Statusmarken für die Situationen können in einer weiteren Tabelle zur Verfügung gestellt und verwaltet werden:

Tabelle 2

situation_id	bool_value	logical_expression	needs_recalculation

In Tabelle 1 werden für jeden logischen Schritt abhängig von auslösendem Signal und Situation die Transitionen gehalten.

Einer der für die Anwendung definierten logischen Schritte ist der aktuelle Schritt, in dem sich die Anwendung userbezogen „befindet“.

Die Liste der Transitionen kann die Veränderung von Standardmarken verschiedener logischer Schritte beinhalten, Werte an den Datenhaushalt der Anwendung übergeben, Ausführung von Funktionen anfordern und zuletzt direkt oder indirekt einen anderen logischen Schritt zum aktuellen logischen Schritt machen.

Mit `parentstep_idref` und `idolstep_idref` kann die Behandlung des Signals an Eltern und Vorbilder weitergereicht werden, wenn

keine der Regeln im aktuellen Schritt greift. Die Anwendung der geerbten Regeln kann auch explizit in der Transitionsliste angefordert werden.

In Tabelle2 werden für jede definierte Situation der aktuelle Wahrheitswert (`true` oder `false`) und die Ermittlungsvorschrift verwaltet. Mit `needs_recalculation` kann intern angezeigt werden, dass der Wahrheitswert neu ermittelt werden muss. Das ist dann notwendig, wenn sich ein Datenwert oder eine andere Statusmarke ändert, die in die Berechnung der aktuellen Statusmarke einfließt. Alle Statusmarken, deren Ermittlungsvorschrift den betroffenen Wert beinhaltet, werden dann ungültig gesetzt. Das wird für alle Statusmarken wiederholt, die wiederum in ihrer Ermittlungsvorschrift eine der auf ungültig gesetzten Statusmarken beinhaltet.

Dieser rekursive Algorithmus erreicht, dass Statusmarken von Situationen nicht bei jedem Abruf, sondern nur bei Bedarf neu ermittelt werden. Trotz des Aufwands für das rekursive Durchsuchen der Tabelle nach abhängigen Werten kann diese Methode die Geschwindigkeit der Anwendung steigern.

Alternativ kann man natürlich auch mit der Holzhammermethode bei Änderung eines Wertes, der in eine beliebige Situationsermittlungsvorschrift einfließt, alle Situationen auf ungültig setzen und so die Neuberechnung aller Statusmarken erzwingen.

Berechnen der Statusmarken

Bei der Berechnung der Statusmarken ist die Reihenfolge der Durchführung von Bedeutung. Die richtige Reihenfolge kann schwierig zu ermitteln sein, wenn nicht klar ist, welche Situationen von welchen abhängen. Ein rekursiver Algorithmus hilft hierbei weiter, kann aber aufwändig sein und kann zirkuläre Abhängigkeiten nicht verhindern.

Es empfiehlt sich daher, eine Hierarchie von Situationen aufzubauen und den Situationsbaum von den Blättern des Baums aufwärts zu berechnen.

Durch hierarchischen Aufbau der Situationen kann auch unkompliziert vermieden werden, dass Situationen zirkulär voneinander abhängen und so zu Dead Locks bei der Berechnung der Statusmarken führen.

9.3 Prozessbeschreibungen im UI abbilden

Wie man eine vorliegende Prozessbeschreibung in eine Prozesslogik für ein User Interface umformt, zeigt der folgende Leitfaden.

- Definieren aller Prozesszustände als logische Schritte
- Anordnung der Schritte in einer Sequenz von der Wiege bis zur Bahre
- Aufbau einer Hierarchie aus logischen Schritten durch Verfeinerung und Generalisierung der Schritte
- Definition von Interaktionen (Transitionen) auf Schrittebene
- Definition der Situationen, auf die bei der Steuerung des Prozesses Bezug genommen wird
- Aufbau der Situationenhierarchie und Formulieren der Ermittlungsvorschrift für die einzelnen Statusmarken
- Auswahl der Schritte, die ein UI haben
- Definition von steuerungsrelevanten Kontrollelementen
- Detaillieren der Interaktionen auf Ebene der Kontrollelemente

Beispiel: Situationen verwenden

Gegeben sei ein Prozess zur Buchung eines Hotels. Er beinhaltet

1. die Suche nach einem Hotel,
2. die Buchungsdatenerfassung und
3. die Buchungsdurchführung.

Die Abläufe, die funktionalen Anforderungen und die Randbedingungen seien in der Prozessdefinition (z.B. in einem Aris-Prozessmodell nebst Begleitdokumentation) beschrieben.

1. Logische Schritte

```
hotel_suchen
buchung_vorbereiten
buchung_bestaetigen
```

2. Ablaufsequenz

```
ort_und_zeitraum_festlegen
hotel_aus_trefferliste_waehlen
kundendaten_erfassen
zahlungsoptionen_festlegen
buchung_ueberpruefen_und_akzeptieren
bestaetigung_drucken
```

3. Hierarchie der logischen Schritte

```
hotel_suchen
  ort_und_zeitraum_festlegen
  hotel_aus_trefferliste_waehlen
buchung_vorbereiten
  kundendaten_erfassen
  zahlungsoptionen_festlegen
buchung_bestaetigen
  buchung_ueberpruefen_und_akzeptieren
  bestaetigung_drucken
```

4. Transitionen auf Schrittebene und Verwendung von Situationen

```
step hotel_suchen
  step ort_und_zeitraum_festlegen
    on step exit
      request search_hotels
  step hotel_aus_trefferliste_waehlen
    on stepexit
      request set_hotel_id
step buchung_vorbereiten
  on step enter if ~situation.hotel_gewaehlt = false
  on step exit ~situation.kundendaten_erfasst = true
  step kundendaten_erfassen
  step zahlungsoptionen_festlegen
buchung_bestaetigen
  step buchung_ueberpruefen_und_akzeptieren
  step bestaetigung_drucken
```

5. Definition von Situationen und Anwendungsdaten

```
situation hotel_gewaehlt = ~datavalue.hotel_id != empty
situation kundendaten_erfasst = false
situation buchungsbereit = ~hotel_gewaehlt & ~kundendaten_erfasst
situation gebucht = ~datavalue.buchungs_id != empty

datavalue hotel_id
datavalue buchungs_id
```

9.4 Ablaufstrukturen visualisieren

Aus den extrahierten Prozessteuerungsdaten können Prozessgraphen und Ablaufmodelle für gängige Modellierungs- und Visualisierungswerkzeuge erzeugt werden.

Zum Beispiel können die Prozesssteuerungstabellen in ein Flussdiagramm in yEd oder in ein UML Activity-Diagramm für MD (Magic Draw UML) transformiert werden. Hilfreich ist dabei, wenn das Zielwerkzeug über automatische Layoutalgorithmen verfügt.

Anderenfalls ist die Transformation in einen Graph aufwendiger, weil dann die Positionen der Knoten des Graphs und unter Umständen auch der Kanten durch den Transformationsalgorithmus berechnet und in die Zieldatei mit ausgegeben werden müssen.

Prinzipiell kann aus den Prozesssteuerungsdaten ein Prozessmodell für ein beliebiges Werkzeug erzeugt werden. Auch die Detaillierungstiefe des erzeugten Prozessmodells kann variiert werden.

9.5 Nutzen

Die Prozessbeschreibung kann aus einem Prozessmodell in ein UI-Modell übernommen werden. Umgekehrt ist auch eine Extraktion und Umformung der Ablaufinformationen aus dem UI-Modell möglich.

Aus den im UI-Modell vorhandenen Ablaufinformationen können etwa Ablaufstrukturgraphen, Workflowbeschreibungen und Steuerungsdaten für Workflowmanagementsysteme extrahiert werden.

Querverweise

Die Formulierung der Prozesslogik ist in der UI-Sprache Lucia in Form von Interaktionen auf der Ebene von logischen Schritten enthalten. Näheres hierzu finden Sie im Kapitel „Sprache“.

Die Ermittlung und Bereitstellung von Statusmarken kann in der UI-orientierten Architektur als Teil des Kontexthandlings verstanden werden und sollte in einer eigenen Softwarekomponente umgesetzt werden. Näheres hierzu finden Sie im Kapitel „Architektur“.

Kapitel 10: Dokumentation

… in dem die Informationen aus dem User Interface-Modell zum Gerüst für Handbücher und für das Hilfesystem werden.

Kapitelziele

- *Handbuch und Hilfe zeitgleich mit der Software liefern.*
- *Automatisierte Übernahme der anleitenden und beschreibenden Teile des UI-Modells in das Gerüst der Benutzerdokumentation.*
- *Automatisches Erzeugen von Screenshots aus der Bildschirmbeschreibung in der UI-Spezifikation.*
- *Informationen über Änderungen am UI automatisiert an Verfasser der Dokumentation liefern (z.B. auf neu hinzugekommene oder entfallene Teile des UI hinweisen).*

Eine Benutzerdokumentation beschreibt, wie die Oberfläche der Softwareanwendung funktioniert. Ein User Interface, das nicht beschrieben ist, gleicht einem Abenteuerspiel. Der Anwender muss raten und rätseln, ob das beobachtete Verhalten der Softwareoberfläche so gewollt oder ein Fehler ist. Funktionalitäten, die nicht offensichtlich sind, bleiben dem Anwender verborgen, bis sie zufällig oder durch systematische Detektivarbeit entdeckt werden. Versehentlich ausgelöst, führen unbekannte Funktionalitäten zu weiteren Irritationen der Anwender.

Es ist daher von Nutzen, wenn bereits mit der ersten UI-Version, die vom Entwicklungsteam bereitgestellt wird (auch dann, wenn diese erste Version die Endanwender niemals sehen werden) eine UI-Dokumentation (oder eine andere Form der Sollverhaltenbeschreibung) mitgeliefert wird. Sie erleichtert den Anwendervertretern den Einstieg, hilft Missverständnisse zu vermeiden und gibt Aufschluss darüber, was man von der Software erwarten kann und was nicht.

A priori und a posteriori Dokumentation

Eine manuell erstellte Dokumentation beschreibt das UI-Verhalten der Anwendung ***nachträglich***. Ein User Interface-Modell, vorausgesetzt, dass es im Projekt ernst genommen wurde, beschreibt ***vorab***, wie die Oberfläche der Softwareanwendung funktionieren soll.

Selbstverständlich haben beide Beschreibungsformen unterschiedliche Schwerpunkte und unterschiedlichen Aufbau. Jedoch enthält die Vorabbeschreibung der Oberfläche im UI-Modell Informationen über Aufbau und Bedienung der Anwendung, die auch in der Benutzerdokumentation von Belang sind.

Die in einem User Interface-Modell enthaltenen Informationen können also für das Erstellen verschiedener Dokumente herangezogen werden.

Bild 81: Aus einem UI-Modell kann man Dokumente ableiten.

Aus einem hinreichend formalisierten UI-Modell kann man zum Beispiel die strukturierenden und die anleitenden Teile automatisiert in ein Gerüst für die zu erstellende Benutzerdokumentation umwandeln. In ähnlicher Weise lassen sich Berichte über den Entwicklungsstand der Oberfläche und Versionsinformationen generieren.

Die aus einem UI-Modell ableitbaren Dokumente lassen sich in drei Kategorien gliedern:

- Entwicklungsdokumente,
- Benutzerdokumente und
- Dokumente zur Änderungsverfolgung.

10.1 Entwicklungsdokumente generieren

UI-bezogene Entwicklungsdokumente sind meist Bestandteil von Designstudien oder Feinkonzepten. Manuell erstellte UI-Design-Dokumente müssen stets auf den neuesten Stand gebracht werden, weil Designentscheidungen auf Basis von ausführbaren UI-Prototypen erfolgen.

Diese Nachdokumentationsarbeit kann eingespart werden, wenn die ausführbaren Prototypen und die UI-Dokumente mit geeigneten Transformationen aus demselben UI-Modell erzeugt werden.

Entwicklungsbegleitende UI-Dokumente können zum Beispiel sein:

- Aufstellungen, die Entwicklern beim Codieren helfen:
  - Liste der Dialogseiten
  - Liste von Kontrollelementen pro Dialogseite
  - Liste von Kontrollelementen pro Kategorie mit Vorkommen-Aufzählung

  - Liste der Aufrufe von Anwendungsfunktionen pro Dialogseite
  - Liste der Aufrufe von Anwendungsfunktionen pro Funktionsgruppe mit Vorkommen-Aufzählung
- Abnahme- und freigaberelevante Projektdokumente:
  - Dialogseitenbeschreibungen,
  - Übersichten über die Ablaufstruktur der Oberfläche,
  - Spezifikationsdokumente, die automatisch aus dem UI-Modell generiert wurden.

10.2 Benutzerdokumente generieren

Das Erstellen von Benutzerdokumenten erfolgt meist in eigenen Teilprojekten. Die Benutzerdokumentation wird dabei auf Basis einer bereits nahe am Endstand funktionsfähigen Software erstellt. In der Projektpraxis kommt die Dokumentation oft später als die Software in den Test und in den Piloteinsatz.

Wertvolle Informationen über eventuelle Brüche in der Bedienungssystematik oder über die Durchführbarkeit der vorgesehenen Anwendungsfälle kommen damit oft zu spät, um in der Software noch vor dem Ersteinsatztermin korrigiert zu werden.

Wünschenswert wäre daher eine Unterstützung des parallelen Erstellens von Software und Dokumentation. Hierzu kann eine automatisierte Auswertung des UI-Modells auf Bedienungshinweise hilfreich sein. Die so erzeugten Dokumentationsgerüste können den Grundstock einer frühzeitigen Benutzerdokumentation bilden.

Gerüst für Benutzerhandbuch und On screen-Hilfe generieren

Denkbar ist auch, dass die aus dem UI-Modell stammenden Anteile der Benutzerdokumentation als Redaktionsbeiträge eingebunden und automatisch aktualisiert werden.

Dokumente für den Anwender der Software sind in der Regel:

- Gerüst für das Benutzerhandbuch
- Gerüst für interaktive Hilfefunktionen der Anwendung

Software soll den Anwender anleiten und helfen, Fehlbedienung zu vermeiden.

Die Dokumentation, insbesondere ihre interaktiven Anteile, ist nicht nur rechtlich gesehen Bestandteil der Software. Eine Softwareoberfläche hat neben der Aufgabe, Funktionalitäten zu erschließen, auch die Aufgabe, den Anwender anzuleiten, und sollte dabei unterstützen, Fehlbedienungen zu vermeiden.

Daraus lässt sich folgern, dass eine Softwareoberfläche erst zusammen mit den interaktiven Hilfefunktionen und mit dem Benutzerhandbuch vollständig ist. Das Generieren von Gerüsten für diese Oberflächenanteile ist ein Schritt, mit dem man dieser Vollständigkeit näher kommen kann.

10.3 UI-Änderungen nachvollziehen

Änderungsdokumentationen sind zum Beispiel für aufeinander folgende Test- und Pilotierungsiterationen hilfreich. Sie ersparen und vereinfachen die mühsame Suche nach den Stellen der Software, die neu geprüft werden müssen.

Durch den automatisierten Vergleich von zeitlich versetzten Ständen eines UI-Modells kann der Aufwand zum Erstellen dieser vielfach von Anwendern, Testern und Managern geforderten Dokumente gesenkt werden. Zudem kann das automatisierte Vergleichen von UI-Ständen die Vollständigkeit einer Änderungsdokumentation sicherstellen.

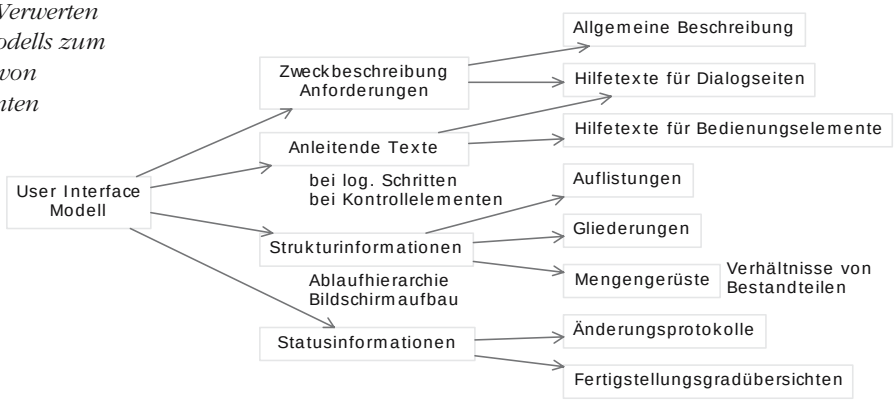
Dokumente, die Änderungen zwischen verschiedenen User Interface-Ständen aufzeigen, können sein:

- Änderungsprotokolle der am UI seit einem bestimmten Datum vorgenommenen Änderungen
- Delta-Dokumentation der Unterschiede zwischen zwei UI-Ständen

10.4 Nutzen

Aus einer in formaler oder in halbformaler Form vorliegenden UI-Spezifikation kann Text und Hypertext für Benutzerhandbuch und für die Bildschirmhilfe erzeugt werden.

Bild 82: Verwerten des UI-Modells zum Erstellen von Dokumenten



Die obige Abbildung zeigt eine Analyse der Informationen in einem UI-Modell aus dem Blickwinkel des Verwertens dieser Informationen zum Erstellen von Dokumentationsteilen.

Das frühzeitige automatisierte Erzeugen des Dokumentationsgerüsts, Generierung von Bildschirmfotos sowie Aktualisierungshinweisen ermöglicht eine starke Optimierung der Arbeiten an der Benutzerdokumentation.

Bei konsequenter automatisierter Verwertung der im UI-Modell enthaltenen, strukturierten Informationen über Zweck und Bedienung der einzelnen Bestandteile bis hin zu einzelnen Kontrollelementen, kann die Erstellung der Benutzerdokumentation und der Entwicklungsdokumente stark beschleunigt werden.

Im Idealfall kann der Softwarehersteller aufeinander abgestimmte und aktuelle Benutzerdokumentation zusammen mit der ersten Testversion der Software liefern. Gleiches kann für Designdokumente erreicht werden.

Die Software könnte damit bereits in frühen Testphasen gegen die Benutzerdokumentation bzw. gegen Designdokumente getestet werden. Ebenso kann durch automatisches Generieren der Dokumente von vorne herein sichergestellt werden, dass die Dokumentation der Spezifikation entspricht.

Das Generieren von Dokumenten hängt eng mit dem Test eines User Interface zusammen, denn anhand von Dokumenten werden die Voraussetzungen für den UI-Test geschaffen werden.

Kapitel 11: Testen

… in dem User Interface-Modelle für den automatisierten Test der UI-Implementierung herangezogen werden.

Kapitelziele

- *Verwerten von Eingabe- und Reaktionsbeschreibungen als Steuerungsregeln für Testszenarien (z.B. automatisch generierte Unit Tests).*
- *Quantitative Aussagen über das UI-Modell anhand der Verteilung der Modellelemente in einer UI-Beschreibung.*
- *Qualitative Aussagen über das UI und das Modell anhand der Erwartungswerte an die Quantitative Verteilung (z.B. ist für jedes Eingabefeld ein Wertebereich festgelegt? Gibt es Kontrollelemente ohne Interaktionen?).*

Testen ist Probefahren nach Plan

Zu testen bedeutet, ein Produkt darauf zu überprüfen, ob es den Erwartungen entspricht. Dazu benötigt man:

- Das Testobjekt (das zu testende Produkt)
- Die Erwartungen (dokumentiertes Sollverhalten)
- Die Testszenarien (Sequenzen von Aktion-Reaktion-Erwartungen)
- Ein Testvorgehen (z.B. Wer testet wann in welcher Testumgebung? Wo werden die Testergebnisse dokumentiert? Was wird mit den Testergebnissen gemacht?)

Eine UI-Beschreibung liefert keine ***expliziten Testszenarien***. Aber sie liefert den Aktionsraum dafür: Wertebereiche, Plausibilisierungen, Verzweigungsmöglichkeiten, Reaktionen auf Eingaben. Diese Information kann verwendet werden, um Testszenarien automatisiert zu generieren, die den oben genannten Aktionsraum prüfen.

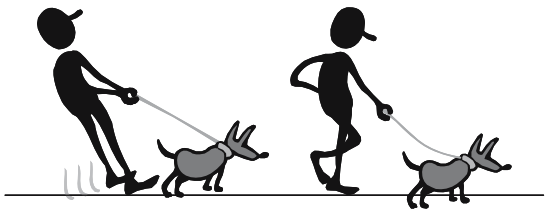


Bild 83: Mit UI-Modellen sind einfache UI-Tests automatisierbar.

Anhand der UI-Modelle können einfache Tests der Oberfläche automatisiert werden. Aus der UI-Spezifikation kann man Testfälle in der Art der im funktionalen Bereich üblichen ***Unit Tests*** generieren.

11.1 Arten von UI-Tests

Man kann zwischen Tests der UI-Erscheinung (Text, Layout, Grafiken) und Tests der UI-Logik unterscheiden.

Das könnten zum Beispiel die folgenden Tests sein:

- Einhaltung von Wertebereichen
- Korrekte Reaktion auf diskrete (im UI erwartete) Eingabewerte
- Korrekte Reaktion auf das Drücken von Schaltern
- Ausführen von erwarteten Dialogseitenübergängen

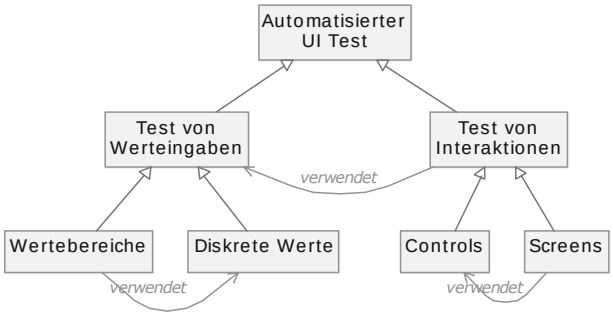


Bild 84: Klassifikation von automatisierten Tests

Für das automatisierte Testen von Wertebereichen und diskreten Werten benötigt man einen Satz von Eingaben, die an die Oberfläche geschickt werden, und einen Satz von daraufhin erwarteten Ausgaben der Oberfläche.

Die von der Oberfläche erwartete Reaktion könnte durchaus sein, dass keine Ausgabe erfolgt, die Oberfläche also z.B. einen Eingabewert, der im definierten Wertebereich liegt, ohne sich zu beschweren annimmt.

Ein elementarer Testzyklus ist ein Kreislauf von Auslöser bis Ergebnis. Das heißt: Der Testzyklus beschreibt, ausgehend von einem bestimmten Punkt im UI, die Auslöser, die zu einem anderen Punkt im UI führen und die dabei vom UI erzeugten Ausgaben.

- System-, umwelt- oder anwenderinitiierte Aktionsauslöser
  + Interpretieren im Kontext
  => Input
- Input + Funktionen ausführen
  => Output
- Output
  + Aufbereitung im Kontext
  + Darstellung
  => Ergebnis

Quantitative Analysen

Quantitative Analysen des UI-Modells stellen statistische Informationen über die Oberfläche zur Verfügung.

Das kann zum Beispiel die Anzahl von Kontrollelementen pro Dialogseite oder die Anzahl von Interaktionen pro Kontrollelement sein.

Quantitative Analysen erfolgen durch Zählen und in Verhältnissetzen von Bestandteilen, zum Beispiel

- Anzahl der verschiedenen Modellbestandteile (z.B. logische Schritte) im Verhältnis zur Modellgröße
- Anzahl der Kontrollelemente pro Dialogbildschirm eines logischen Schritts
- Anzahl der Modellbestandteile pro Modellierungsiteration (Wieviele Modellbestandteile sind auf welchem Fertigstellungsstand gekennzeichnet?)
- Mengengerüst: Übersicht der Modellbestandteile nach Perspektiven (Aussage über die Gleichmäßigkeit des Modells)

Qualitative Analysen

Anhand von Erwartungswerten an Quantitäten und anhand der tatsächlichen Quantitätsverhältnisse können Qualitätsverbesserungspotentiale in der Oberfläche identifiziert werden.

Qualität wird anhand des Eindrucks beurteilt, die einzelne Produktteile und das Gesamtprodukt auf den Tester machen. Qualitative Analysen können helfen, die Qualitätsbeurteilung nachvollziehbarer zu machen.

Zum Beispiel könnte die Regel gelten, dass auf einer Dialogseite zwischen 15 und 35 Kontrollelemente erwartet werden. Die qualitative Analyse würde auf Dialogseiten hinweisen, die nicht in diesem Bereich liegen und daher möglicherweise unvollständig (< 15) oder zu komplex (> 35) sind.

Weitere Beispiele für qualitative Analysen sind:

- Konsistenzchecks (ist das Modell in sich geschlossen?)
- Redundanzchecks (sind gleiche Sachverhalte nur einmal definiert, oder werden Definitionen wiederholt?)
- Layoutchecks (ist das Layout der Dialogseiten konsistent?)

Ebenfalls aussagekräftig ist das Aufzeichnen der zeitlichen Veränderungen von qualitativen Merkmalen und ihre Darstellung als Kurve auf der Zeitachse. Man kann auf diese Weise Entwicklung und Tendenz der Qualität einschätzen.

11.2 Testszenarien generieren

Aus den Informationen in einem UI-Modell können keine fachlichen Testszenarien generiert werden. Fachliche Testfälle sind in der Beschreibung des UI in der Regel nicht enthalten und können aus ihr nicht ausgeleitet werden.

Die UI-Spezifikation unterscheidet sich von einer Testfallspezifikation: Die Testfallspezifikation beschreibt konkrete ***Durchläufe*** und die dabei erwarteten ***Ergebnisse***; die UI-Spezifikation beschreibt hingegen den ***Möglichkeitenraum*** für abstrakte Durchläufe und den ***Erwartungsraum*** der Eingaben.

Für das automatische Testen der Oberfläche können einfachere Tests auf der Ebene einzelner Eingabewerte und Reaktionen auf das Betätigen einzelner Schaltflächen aus der UI-Spezifikation herausanalysiert werden.

Das folgende Diagramm zeigt eine Vorgehenskarte für solche automatisierten UI-Tests.

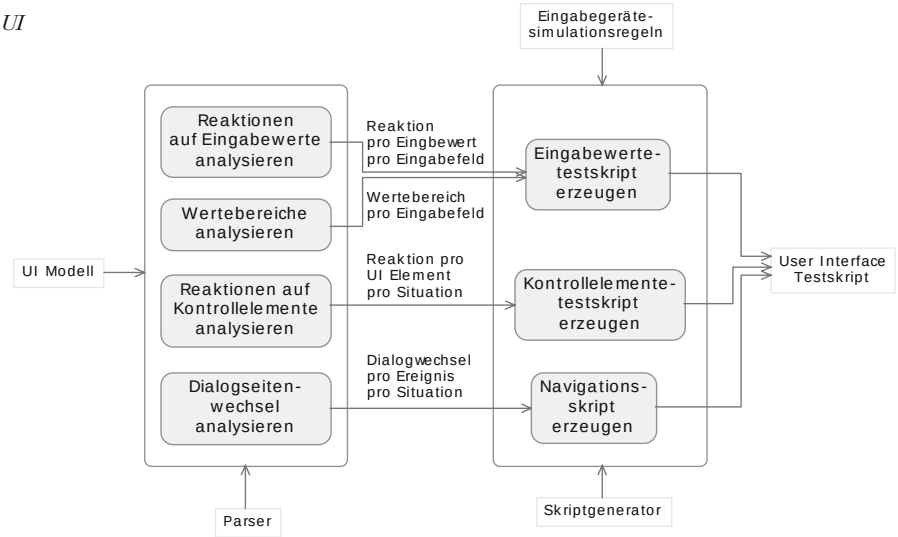


Bild 85: Vorgehenskarte für das automatisierte Testen des UI

Das Testskript kann z.B. pro Eingabefeld automatisiert Werte innerhalb und außerhalb des gültigen Wertebereichs eingeben und die erwartete Reaktion überprüfen, z.B. eine Fehlermeldung nach einem falschen Eingabewert.

- Das Eingabewerteskript testet Wertebereiche und besondere Eingabewerte.
- Das Kontrollelementeskript testet Signale an Kontrollelemente.
- Das Navigationsskript testet die Übergänge zwischen den Dialogseiten.

11.3 Navigation beim UI-Test

Dem Testskript muss bekannt sein, wie man im UI zur betreffenden Dialogseite und zu dem zu testenden Eingabefeld navigiert. Deshalb muss das Testskript mindestens ein Eingabegerät, das zum Bedienen der Oberfläche verwendet wird, kennen, und es muss dieses Gerät simulieren können.

Beim UI-Test muss das Eingabegerät bekannt sein

Das Testskript sendet Eingabezeichen – zum Beispiel Tastendrücke und Zeigegerätbewegungen – an das Eingabegerät, um zu den zu testenden Dialogseiten und Eingabefeldern zu gelangen.

Das Simulieren der Navigation ist gegenüber dem eigentlichen Test von Eingaben und Reaktionen der komplexere Teil des Testskripts. Es empfiehlt sich, hierfür einen allgemeinen Testsequenzer zu erstellen oder den Sequenzer eines Testtools (z.B. MQC (Mercury Quality Center)) zu verwenden und abstrahierte Navigationsanweisungen an diesen zu senden.

Solche Navigationsanweisungen, die der Testsequenzer verstehen sollte, können zum Beispiel „NextField“, „PreviousField“ oder „NextDialog“ sein. Der Aufwand für das Erstellen eines Testsequenzers ist hoch, jedoch die einzige Möglichkeit, das Testskript automatisch ablaufen zu lassen.

Eine weniger effektive Alternative ist das halbautomatische Testen der Oberfläche. Dabei navigiert sich das Testskript nicht selbst durch die Oberfläche, sondern gibt dem Anwender Navigationsanweisungen. Das Testskript weist die testende Person an, zu einer bestimmten Dialogseite und zu einem bestimmten Feld zu navigieren und führt die Tests des Felds erst nach Bestätigung seitens des Anwenders aus.

Das Navigieren zu einer bestimmten Dialogseite und zu einem bestimmten Kontrollelement auf dieser Seite kann durch eine Hintertür in der Oberflächenimplementierung unterstützt werden.

Zum Beispiel kann die implementierte Oberfläche Testanweisungen der Art „`GoTo <ScreenId> [<ControlId>]`“ unterstützen.

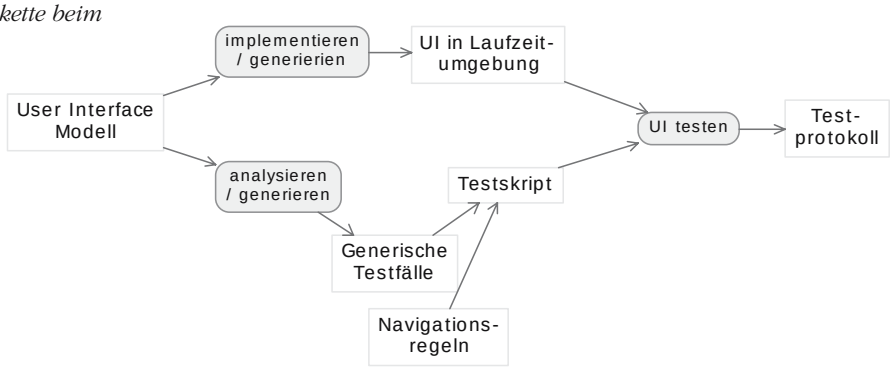
Falls die Oberflächenimplementierung eine solche Navigationshintertür bereitstellt, kann das Testskript für den Oberflächentest ohne die Simulation der Navigation mit Eingabegeräten und ohne Navigationsanweisungen an den Anwender ausgeführt werden. Ohne eine Navigationshintertür können Plausibilitätsprüfungen der UI-Implementierung das Betreten einer Dialogseite bzw. eines Eingabefelds verhindern, z.B. weil die fachlichen Voraussetzungen fehlen, und den Test von Eingabewerten vereiteln.

Unterstützt die UI-Implementierung keine Hintertür für die Navigation mittels eines Testautomaten, dann sind die Tests des Testskripts nur in Verbindung mit echten fachlichen Testszenarien anwendbar. In diesem Fall können die generierten Testanweisungen als Datenquelle für manuell zu erstellende Testfälle von Nutzen sein.

11.4 Nutzen

Anhand einer formalen oder einer halbformalen UI-Spezifikation können automatisierte Massentests der Oberfläche generiert werden. Diese Tests ersetzen keine fachlichen Testszenarien, ersparen aber den Anwendern bzw. den Testern den in der Regel langwierigen und eintönigen Test des Standardverhaltens einzelner Dialogelemente.

Bild 86: Artefaktekette beim UI-Test



Aus dem UI-Modell werden generische Testfälle (z.B. Eingaben innerhalb und außerhalb des definierten Wertebereichs eines Eingabefelds) ermittelt und ein Testskript erzeugt, mit dem das UI in der Laufzeitumgebung getestet wird.

Die Testergebnisse können zur Identifizierung und Behebung von Flüchtigkeitsfehlern an der Oberfläche (zum Beispiel falsch eingestellte Wertebereiche von Eingabefeldern, vergessene Ausgaben) verwendet werden.

Kommen Ihnen solche Peanuts, die einen lange vorbereiteten Test wie ein Kartenhaus umwerfen können, bekannt vor? Kleine, eigentlich leicht zu behebende Oberflächenfehler führen oft genug dazu, dass ein bereits angelaufener Fachtest abgebrochen werden muss.

Mit automatisierten Tests im Vorfeld des Testens von fachlichen Anwendungsszenarien durch Anwender kann die Qualität der Oberfläche für den Test erheblich gesteigert werden. Automatisierte Oberflächentests helfen, beim Testen die wertvolle Arbeitszeit der Test- und Fachspezialisten gezielt einzusetzen und Kosten einzusparen.

Kapitel 12: Projektmanagement

… in dem UI-Modelle in das organisatorische Regelwerk des Projekts integriert werden.

Kapitelziele

- *Klare Zuständigkeit und Mitwirkungspflichten für UI-Modelle.*
- *Nachvollziehbare Checkpoints, Schnittstellen und Synchronisationspunkte zwischen den Projektrollen.*
- *Wissen, wie man bei der Integration UI-orientierter Methoden in die Projektorganisation vorgeht.*
- *Risiken (z.B. Kompetenzüberschneidungen und Abschieben von anderen Entwicklungsaufgaben auf die UI-Modellierung) erkennen und diese einschränken.*

Fast jeder fühlt sich kompetent, bei der Benutzeroberfläche einer Software mitzureden und mitzugestalten, aber fast niemand möchte das gesamte User Interface verantworten.

Bild 87: Beim UI fühlen sich viele kompetent.

Sorgfältig aufgebaute UI-Modelle besitzen eine große Aussagekraft. Mit ihrer Hilfe werden verschiedene Bereiche der Entwicklung im Projekt transparent und vergleichbar gemacht.

Zugleich wird der Stand des Entwurfs von allen Projektbeteiligten messbar, weil man keine Spezialkenntnisse mehr benötigt, um anhand der Oberfläche zu sehen, wie weit der Entwurf gediehen ist. Diese Wirkung von UI-Modellen stellt die Organisation des Projekts vor besondere Herausforderungen.

Zum einen muss vermieden werden, dass das User Interface zum Sündenbock wird. Ein UI-Modell kann andere Entwicklungstätig-

keiten nicht ersetzen. Daher kann man ein fehlerhaftes Datenmodell oder fehlende Funktionen nicht mit Unvollständigkeiten in der UI-Beschreibung begründen. Dies zu kommunizieren und sicherzustellen, liegt in der Verantwortung des Projektleiters.

Andererseits muss vermieden werden, dass das UI-Modell ausgegrenzt wird. Ein UI-Modell liefert Informationen über Zusammenhänge und begünstigt die Evolution der Anwenderforderungen. Auf diese Weise werden Lücken und Fehler in der Konzeption der Anwendung sichtbar, was die Entwickler zur fortlaufenden Korrektur der Funktions- und Datenmodelle zwingt. Als Abwehrreaktion auf diese fortwährende Störung können UI-Modelle in den Verruf geraten, ständig alles zu verändern und die Realisierung der Anwendung aufzuhalten.

12.1 Arbeiten im Kontext der UI-Anforderungen

Die Anforderungen an das User Interface liefern Informationen, die alle Teile der Software betreffen. Diese Informationen müssen an alle Projektbeteiligten verteilt werden. Umgekehrt werden für das Entwickeln der Oberfläche Informationen über alle Teile der Software herangezogen und verwendet.

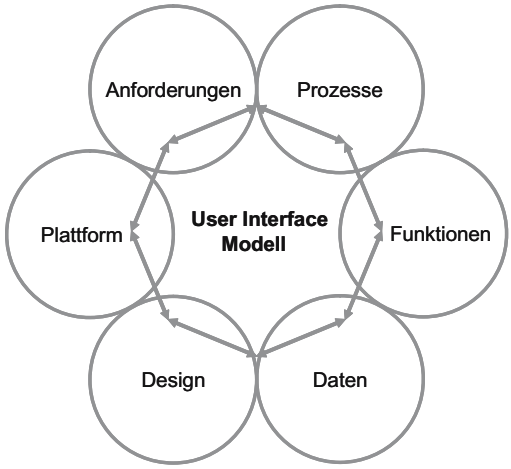


Bild 88: User Interface-Entwicklung ist ein interdisziplinäres Thema

Die Anforderungen an das User Interface sind Kerninformationen für das ***Wissensmanagement*** des Projekts. Sie können verwendet werden, um nutzbringend das gemeinsame Vorgehen im Projekt zu koordinieren.

Informationen über Prozesse werden verwendet, um die Ablaufstruktur des UI zu formen. Informationen über Funktionen werden dazu herangezogen, diese Funktionalität per UI dem Anwender zur Verfügung zu stellen. Informationen über Daten

werden zur Steuerung von Funktionen und Abläufen benutzt. Die Informationen über das Design bestimmen das Look und Feel der Software. Die Informationen über die Zielplattform geben Einschränkungen und Rahmenbedingungen für die vom UI angebotenen Darstellungs- und Bedienungsformen.

12.2 UI-orientiertes Vorgehen

Die Entwicklung der Oberfläche sollte im Entwicklungsprozess als ein eigener, zur Funktionsentwicklung paralleler Entwicklungsbereich angesiedelt und etabliert werden.

Es ist die Aufgabe des Projektleiters, die für das UI Verantwortung tragenden Personen zu stärken und die Mitwirkung anderer Rollen am UI-Entwurf einzufordern. In einem Projekthandbuch sollte klar festgelegt sein, wer was zum UI beitragen muss und woran die Qualität dieser Beiträge gemessen wird.

UI-Modelle zeigen Zielkonflikte auf

UI-Modellierung ist ein Querschnittsthema. Sie vernetzt und verbindet Verantwortungsbereiche und kann deshalb Zielkonflikte offen legen, z.B. zwischen Datenmodellierern und den Befürwortern einer detaillierten Historie aller Eingaben im UI.

Der Projektleiter muss deshalb Kommunikationsprozesse fördern und eine Instanz für die laufende Beilegung der aufgedeckten Widersprüche und Zielkonflikte stellen.

Checkliste für UI-orientiertes Vorgehen:

- UI-Entwurf als eigenständiges Artefakt im Entwicklungsprozess aufstellen.
- Informationsschnittstelle zwischen UI-Entwurf und funktionalen Entwürfen festlegen.
- Regelmäßigen Abgleich zwischen UI-Entwurf und Funktionsmodellen etablieren.
- Rollenkompetenz definieren und einhalten.
- Zielkonflikte identifizieren und auflösen.
- Checklisten für Erreichen von Fertigstellungsgraden des UI-Entwurfs verwenden.

12.3 Integration sicherstellen

Wechselnde UI-Anforderungen verunsichern die Entwickler

Neue oder veränderte Anforderungen an das User Interface können eine bereits begonnene technische Lösung in Frage stellen. Die Anwendervertreter und „ihr“ User Interface entscheiden aus Sicht der Entwickler scheinbar willkürlich über die Technik. Entwickler mögen das nicht. In der Vorstellung der Entwickler

beschneidet es ihren Gestaltungsspielraum und gefährdet die Ergebnisse ihrer Arbeit.

Oberflächen - so argumentieren Entwickler oft - kann man eigentlich erst entwickeln, wenn die Funktionsanalyse abgeschlossen ist, sonst müssen sie ständig geändert werden. Zu dieser Argumentation gehört auch die weit verbreitete Vorstellung, dass Oberflächen nur ein Aufsatz auf die wesentlich komplexere Funktionalität sind.

Vermeiden des Ausgrenzens

Um den Sündenbock- wie den Ausgrenzungseffekt zu vermeiden, müssen im Projekt klare Schnittstellen und Synchronisationspunkte zwischen dem sich schnell ändernden UI-Modell und den weniger agilen Artefakten wie Datenmodellen, Klassen und Programmcode festgelegt werden.

Ebenso muss die Mitwirkung der verschiedenen Rollen am Vervollständigen des User Interface-Modells und am Verwerten der Informationen aus dem UI-Modell klar geregelt sein und kontinuierlich durchgesetzt werden.

Wird die Oberfläche nicht fest im Projekt verankert und ins Vorgehen integriert, dann wird sie unter Umständen ausgegrenzt, und die Realisierung der Software wird verschiedenen Risiken ausgesetzt.

Beispiel: Oberfläche wird gerne ausgegrenzt

Beispiele für negative Folgen des Ausgrenzens der Oberfläche aus dem Entwicklungsvorgehen.

Bild 89: Funktions- und Daten-Entwickler grenzen das User Interface gerne aus

Szenario 1:

Framework wird ohne Standard-UI entwickelt

Ein Softwarehersteller entwickelt ein Banking Framework. Als Entwicklungsziel wird die Bereitstellung einer Mittelschicht für die Abwicklung von Fachfunktionen gesetzt.

Kritische Projektentscheidung:

- Eine Standardoberfläche ist für das Framework nicht vorgesehen.
- Das Argument für diese Entscheidung ist, dass für jeden Kunden individuell ein seinen Bedürfnissen entsprechen-

des User Interface auf die Frameworkfunktionen aufgesetzt werden soll.

Folgen:

- Das Produkt ist vor dem Umsetzen einer ersten Referenzlösung nicht vorführfähig. Der erste Kunde muss eine „Katze im Sack“ kaufen. Der nächste Kunde muss von der Individuallösung des Referenzkunden auf den Umfang des Standardframeworks abstrahieren können und wollen.
- Das Produkt kann nur auf Ebene einzelner Funktionen, aber nicht auf Use Case-Ebene getestet werden.
- Die Vollständigkeit und Anwendbarkeit des Frameworks kann erst beim ersten Kunden reifen.
- Da für Interessenten die Lösung nicht erlebbar ist, wird das Produkt nicht erworben. In der Folge wird das Projekt eingestellt.

Szenario 2:

Use Cases gehen über die Grenzen von beauftragten Komponenten

In einem Großprojekt wird die Softwareunterstützung für eine integrierte Konstruktions- und Beschaffungsumgebung eines Automobilherstellers geplant und umgesetzt. Die Konzeption erfolgt nach Anwendungsfällen und beinhaltet auch UI-Entwürfe.

Kritische Projektentscheidung:

- Die Umsetzung wird nach technischen Komponenten gegliedert beauftragt.
- Das Argument für diese Entscheidung ist das Senken der Komplexität einzelner Lösungsanteile durch die Orientierung an den beteiligten technischen Systemen.

Folgen:

- Die Integration der Lösung auf Use Case-Ebene liegt im „Niemandsland“ zwischen den plattformgetriebenen Lösungsteilen.
- Die konzipierten Benutzeroberflächen werden in den Lösungsteilen zum Teil neu entworfen. Das Übergehen der Konzepte erfolgt oft unter Hinweis auf Scope und auf die technischen Möglichkeiten der jeweiligen technischen Plattform.
- Die vom Softwarehersteller programmierten Oberflächen sind nicht vorab bekannt. Sie stehen erst bei Lieferung der Software fest. Tests sind daher nur schwer planbar.
- Beim Test wird festgestellt, dass die gelieferten Funktionalitäten nur lückenhaft zu den konzipierten Anwendungsfällen und zu den abzubildenden Prozessen passen.

- Da nur einzelne Funktionen, aber nicht durchgängige Prozesse zur Verfügung stehen, verzögern sich Test und Pilotierung der Lösung. In der Folge überschreitet das Projekt erheblich den Zeit- und Budgetrahmen. Zudem sind die Anwender vom Ergebnis enttäuscht, das nicht dem Konzept und damit nicht den Erwartungen entspricht.

Szenario 3:

Dynamisches Regelwerk wird starr codiert

Ein IT-Consultingteam wird mit der Erstellung einer Projektvorstudie für eine Software beauftragt, die Kreditengangements unterstützen soll. Der Auftrag beinhaltet die Prozess- und die Anforderungsanalyse sowie die Erstellung eines auf die Ergebnisse abgestimmten User Interface-Prototypen. Die Navigation und die Bildschirmmasken des Prototypen werden maschinell aus den Ablaufbeschreibungen und den Anforderungsdaten generiert.

Kritische Projektentscheidung:

- Der Kunde beauftragt die Umsetzung der Anwendung auf Basis des User Interface-Prototypen in einer neuen Architektur. Dazu merkt er an, dass er einen geringen Aufwand erwartet, da die Anwendung durch den ausgereiften Prototypen so gut wie fast fertig sei.
- Der mit der Umsetzung beauftragte Softwarehersteller verwirft den generativen Ansatz und erstellt eine zielplattformspezifische Spezifikation der Ablaufsteuerung und der Bildschirmmasken.
- Das Argument für diese Entscheidung ist, dass die im Prototyp erlebbare Bedienung auf der Zielplattform nicht eins zu eins umsetzbar sei.

Folgen:

- Während die Datenbasis des UI-Prototypen vom Kunden weiter verfeinert wird, findet die Realisierungsplanung auf Basis der Ergebnisdokumente der Studie statt. Die Konzeption und die Realisierungsplanung driften auseinander.
- Die Entwickler sehen sich wechselnden Anwenderforderungen ausgesetzt und kommen mit der Codierung dem sich ständig ändernden Prototypen nicht hinterher.
- In der Zielarchitektur wird die Ablauflogik starr codiert. Die gewünschte Flexibilität der Präsentations- und Ablaufregeln kann vom Hersteller in der Zielarchitektur nicht umgesetzt werden. In Folge wird das Projekt wegen Unvereinbarkeit von Architektur und Konzeption eingefroren und später mit einem anderen Hersteller neu konzipiert und realisiert.

Die obigen Szenarien sind an wirkliche Projektsituationen angelehnt. In der Folge haben diese Projekte den vorgesehenen Zeit- und Budgetrahmen erheblich überschritten. Nach meinem Dafürhalten hätte die stärkere Ausrichtung an User Interface-Entwürfen diese Projekte erfolgreicher machen können.

Die schon erwähnte Integration des UI-Entwurfs in das Entwicklungsvorgehen spielt dabei eine große Rolle. Besonders wichtig ist eine klare Aufteilung von Rollen und Aufgaben zwischen den Entwicklern des UI und der funktionalen Anwendung (vgl. Kapitel „Funktionsmodellierung“) sowie zwischen den einzelnen an der UI-Entwicklung beteiligten Personen.

12.4 Rollen und Aufgabenteilung bei der UI-Entwicklung

Das folgende Diagramm zeigt die typischerweise an der Entwicklung einer Oberfläche beteiligten Hauptrollen.

Das sind:

- Anwendervertreter („Konsumenten“)
- Spezifizierer („Bauplaner“)
- Architekt („Bauleiter“)
- Programmierer („Umsetzer“)
- Projektleiter (Koordinator, Eskalationsstelle)

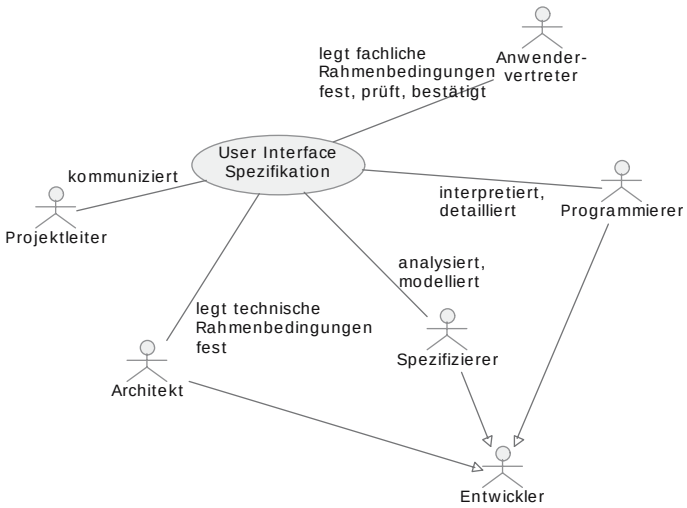


Bild 90: UI-bezogene Rollen und Aufgaben im Entwicklungsprozess

Das Entwicklerteam (Programmierer, Spezifizierer, Architekt) folgt bei der Entwicklung der Oberfläche den von den Anwendervertretern festgelegten Rahmenbedingungen. Der Fortschritt

Ihrer Arbeit kann am Erfüllungsgrad der Anwenderforderungen gemessen werden.

Der Architekt legt die technischen Rahmenbedingungen der Entwicklung fest.

Der Projektleiter legt die organisatorischen Rahmenbedingungen der Entwicklung fest und kommuniziert die Arbeitsergebnisse an die Projektgremien.

Weitere an der UI-Entwicklung beteiligte Spezialisten können sein:

- Designer / Grafiker
- Handbuchautor
- Tester / Qualitätsmanager
- Usability-Experte / Bedienkonzeptentwickler
- Technik- und Business Consultants
- Prozessspezialisten
- Integratoren von Standardkomponenten

Eine detaillierte Landkarte der am Bau einer Oberfläche beteiligten Rollen, ihrer Beziehungen und Aufgaben kann folgendermaßen aussehen:

Bedienkonzeptentwicklung („Navigation“)

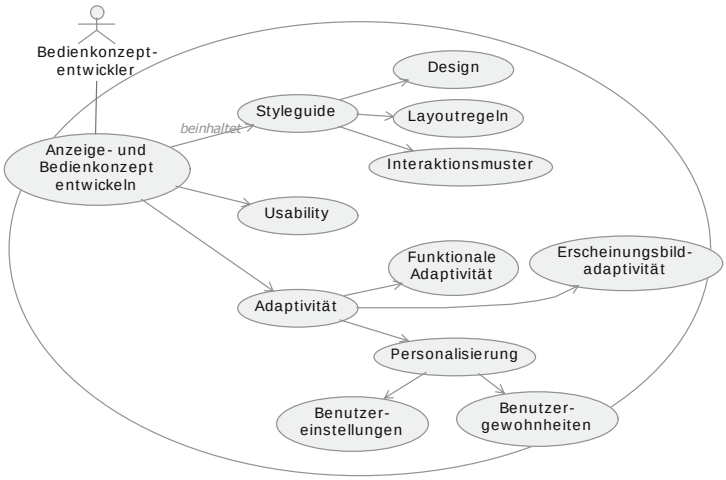


Bild 91: Use Cases Map der UI-Entwicklung - Bedienkonzept

Bedienkonzeptentwickler entwickeln Grundregeln für die Handhabung der Software (Bedienungsmetaphern). Hierzu gehören z.B. die Usability, das UI-Styleguide und Regeln für die Anpass-

barkeit der Benutzeroberfläche zur Konfigurationszeit und zur Laufzeit.

Erscheinungsbildentwicklung („Grafikdesign“)

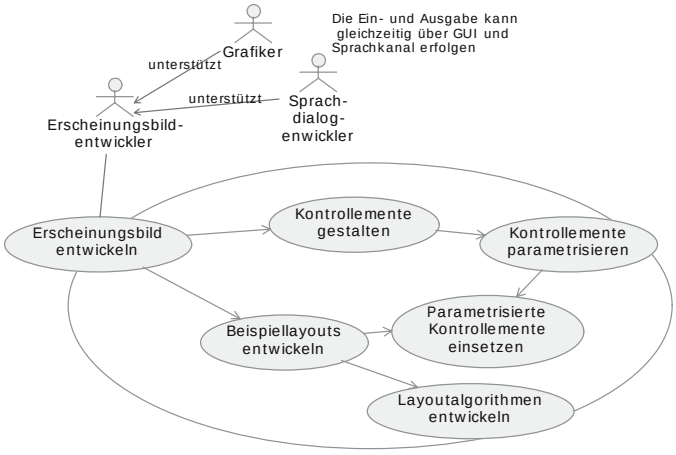


Bild 92: Use Cases Map der UI-Entwicklung - Erscheinungsbild

Erscheinungsbildentwickler entwickeln, wie die Software aussieht und/oder sich anhört. Hierzu gehört z.B. das Grafikdesign oder allgemein gesagt das „Look and Feel“ der Software.

Struktur- und Verhaltensentwicklung („Logik“)

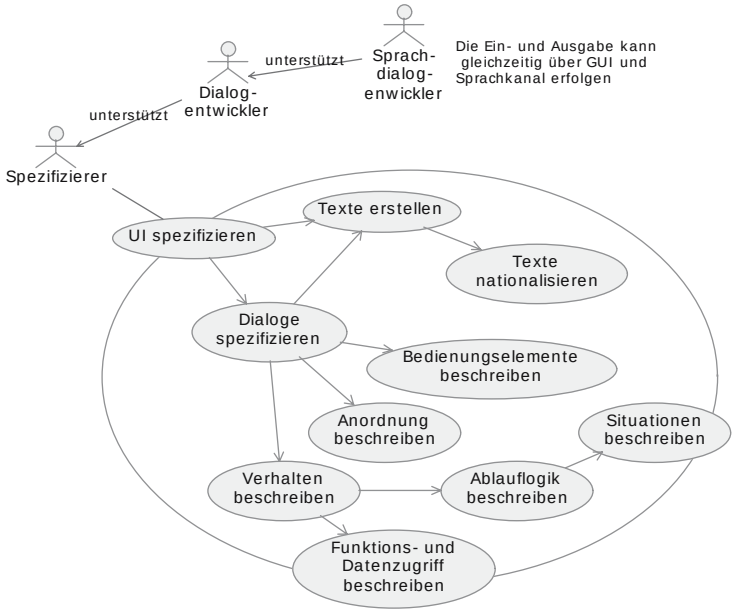


Bild 93: Use Cases Map der UI-Entwicklung - Inhalte

Spezifizierer beschreiben die Inhalte und das Verhalten der Software. Hierzu gehören z.B. die Reaktionen auf einzelne Eingaben, aber auch die Reihenfolge der Dialogseiten.

Methodenentwicklung („Modellmanagement“)

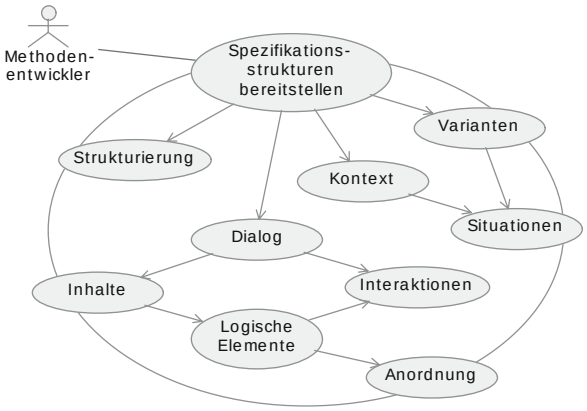


Bild 94: Use Cases Map der UI-Entwicklung - Methoden

Methodenentwickler pflegen die Systematik der UI-Beschreibung. Hierzu gehören z.B. Erfassungsschablonen für Dialogseitenbeschreibungen.

Technische Umsetzung („Implementieren“)

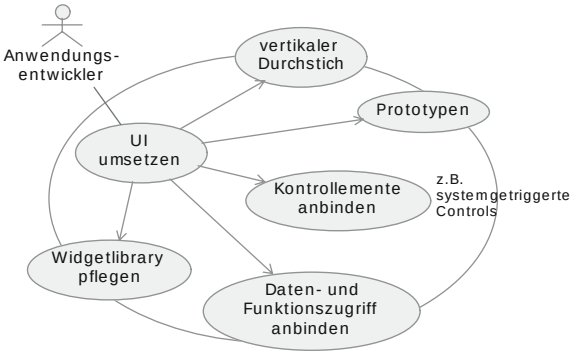


Bild 95: Use Cases Map der UI-Entwicklung - Umsetzen

Anwendungsentwickler setzen das UI um. Hierzu gehört z.B. die Anbindung der UI-Logik an die interne Anwendungslogik und an den Datenhaushalt der Anwendung.

UI-Test („Prüfen“)

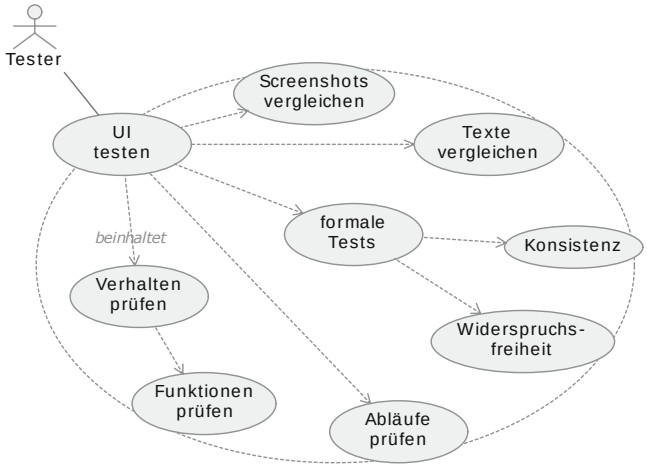


Bild 96: Use Cases Map der UI-Entwicklung - Testen

Tester prüfen die Übereinstimmung von UI-Implementierung und UI-Spezifikation. Hierzu gehört z.B. das Nachvollziehen einzelner Interaktionen, aber auch das Vergleichen von Bildschirmen mit Entwürfen.

Die UI-Entwicklung umfasst die Definition bzw. Beschaffung folgender Informationen, die in ein Gesamtmodell einfließen.

- Dialogseitenstruktur und Inhalte der Dialoge [Logische Schritte]
- Logische Dialogelemente [Listen, Schaltflächen, etc.]
- Ablaufhierarchie [Menübaum, Schrittübergänge]
- Ablauflogik [Ereignisse, Bedingungen, Reaktionen]
- Schnittstelle zur Anwendungslogik [Situationen, Funktionen, Zustände von angeschlossenen Geräten]
- Wertebereiche [Datenmuster, Listen]
- Text- und Grafikressourcen [Anzeigetexte, Grafiken]
- Anforderungsbeschreibung [Prosabeschreibung]
- Variantendefinition und Variantenbezug [Konfiguration]
- Modellkonfiguration (Anzeige- und Bedienelemente, Variantenbeschaltung, Stilelemente, Skins)

Der UI-Entwicklungsprozess kann in folgende Tätigkeiten unterteilt werden:

- Grundkonzepte erstellen
- Anforderungen verwalten und laufend mit Fahrzeugfunktionen abstimmen

- Anforderungen modellieren
- Modellierte Anforderungen analysieren [mittels erstellter Visualisierungen]
- Optional: UI der Anwendung simulieren [Prototyping, ausführbare Spezifikation]
- Optional: Zielcode für eine UI-Umsetzung generieren

Vergleichen Sie dazu auch die Aufteilung des UI-Entwicklungsprozesses in Iterationen (siehe Abschnitt „Detaillierungsstufen“ im Kapitel „Detaillieren“). In jeder UI-Entwicklungsiteration können alle oben aufgezählten Tätigkeiten vorkommen.

Folgende Rollen sind am User Interface-Entwicklungsprozess beteiligt:

- Entscheider
  - Auftraggeber
  - Projektleiter
  - Architekt
- Entwickler
  - Anforderungsingenieur, Spezifizierer
  - Prototypenbauer, Werkzueugbauer
  - Anwendungs- und Systementwickler
  - Qualitätsingenieur, Qualitätsmanager
  - Fachspezialist, Anwendervertreter

Tool Chain

Zur Unterstützung des UI-Entwicklungsprozesses sollte die Verfügbarkeit folgender Werkzeuge bzw. Funktionseinheiten sichergestellt werden.

- Modellverwaltungswerkzeug
- Modelleditor (ggf. mehrere spezialisierte Editoren)
- Modellauswerter, Modellvisualisierer
- UI-Simulator mit Styleguide-Anbindung, d.h. in dem durch das aktuelle Anzeige- und Bedienkonzept definierte Look und Feel
- Zielcode Generator mit Beispielapplikation auf der Zielplattform

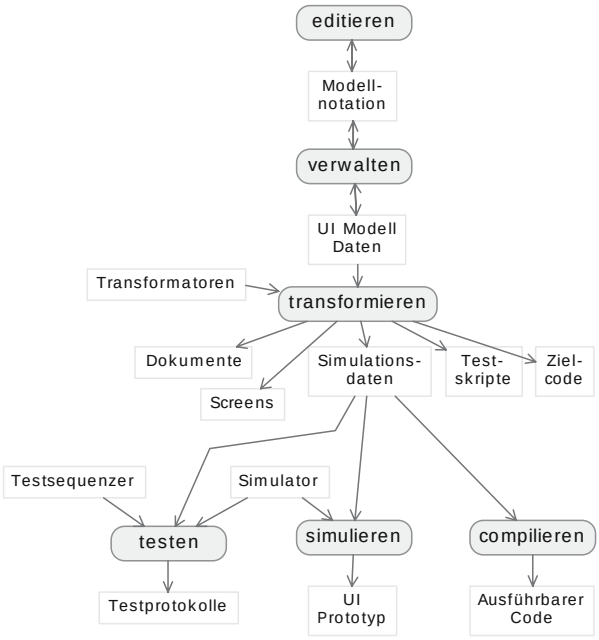


Bild 97: Tool Chain für die UI-Entwicklung

Die neuen Funktionen sollten in die bestehende Methoden- und Werkzeuglandschaft integrierbar sein.

12.5 Nutzen

UI-Modelle erhöhen die Transparenz im Projekt und machen die Grenzen zwischen den Aufgaben weicher.

Bei der Einführung der UI-Modellierung muss der Projektleiter daher besonderen Wert auf die Festlegung und Einhaltung von Zuständigkeiten und Mitwirkungspflichten legen.

Empfohlene Projektmanagement-Aktivitäten:

- Arbeiten im Kontext der UI-Anforderungen
- Sicherstellen der Integration der UI-Entwürfe in die Aktivitäten aller Entwickler
- Klare Aufgabenteilung zwischen UI-Entwicklern und Funktionsentwicklern
- Klare Aufgabenteilung zwischen verschieden spezialisierten UI-Entwicklern

Durch ein aktives, auf Offenlegung und Auflösung von Konflikten ausgelegtes Projektmanagement wird sichergestellt, dass die

UI-Modellierung als Kommunikations- und Abstimmungsplattform dient und nicht als Abladehalde für unbequeme Fragestellungen der Entwicklung benutzt wird.

So wird wiederum vermieden, dass das UI-Modell von den Entwicklern auf ein Abstellgleis gestellt wird.

Ein so aufeinander abgestimmtes und koordiniertes Vorgehen kann helfen, Entwicklungslaufzeiten erheblich zu verkürzen und ein stabiles, erwartungskonformes Projektergebnis sichern.

Kapitel 13: Lebenszyklusmanagement

… in dem plattformunabhängige User Interface-Modelle die Wiederverwendung und kontinuierliche Weiterentwicklung von bewährten Oberflächen ermöglichen.

Kapitelziele

- *Oberflächen nicht immer wieder wegwerfen und neu entwickeln, sondern kontinuierlich fortentwickeln.*
- *Bei Technologiewechsel und bei Reengineering Qualität der Oberfläche auf Basis bewährter Eigenschaften weiterentwickeln.*
- *Neuentwicklungskosten senken, Investitionswert erhalten.*

Life Cycle Management

Im Laufe der Zeit können aus hilfreichen Anwendungen (vor allem im Verwaltungsbereich) durch gewachsene Erweiterungen und durch so genannte Papierschnittstellen Instrumente der Systembefriedigung werden. Bei so wild gewachsenen Anwendungen können die Anwender nicht mehr den Nutzen für ihre tägliche Arbeit und für das Unternehmen erkennen.

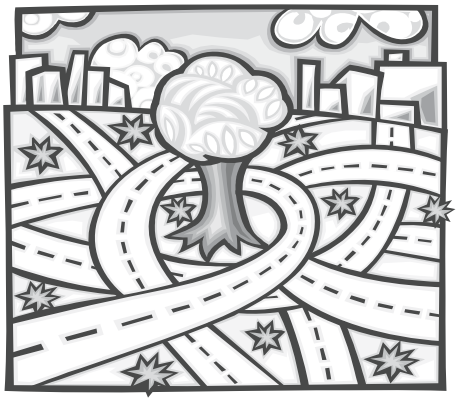


Bild 98: Gewachsene Anwendungen werden immer weniger überschaubar

Lebenszyklusmanagement kann helfen, solche Fehlentwicklungen zu vermeiden. Das Risiko von Softwarebausteinen, die an Fachprozessen vorbei arbeiten, und das Risiko von technologischen Sackgassen sollen dabei minimiert werden.

Lebenszyklusmanagement hat das Ziel, den Einsatz einer unternehmensweiten Softwarelösung über lange Zeiträume hinweg so zu planen und zu organisieren, dass Paradigmenwechsel und

technologische Änderungen rechtzeitig erkannt und berücksichtigt werden.

User Interface-orientiertes Lifecycle- und Cross-Lifecycle-Management kann besonders bei folgenden Situationen zum Senken dieser Risiken, zum Vermeiden von Verwendbarkeitsverlusten und zum Erhalt des Werts einer Softwarelösung beitragen.

- Änderungsanforderungen (Change Requests).
- Release-Planung; stufenweise Realisieren von Features
- Technisch bedingte Migration der Softwarelösung in eine andere Laufzeitumgebung.
- Ersetzen einer bestehenden Lösung durch eine Standardlösung; i.d.R. mit Customizing (kundenindividuelle Konfiguration, Anpassung und Erweiterung) der Standardlösung.
- Ersetzen einer bestehenden Lösung durch Reengineering auf einer anderen technischen Plattform.
- Austausch des Front Ends: Kapselung des Altsystems (i.d.R. Hostanwendung) als Serviceschicht für ein neues User Interface (z.B. ein Thin Client im Web Browser)

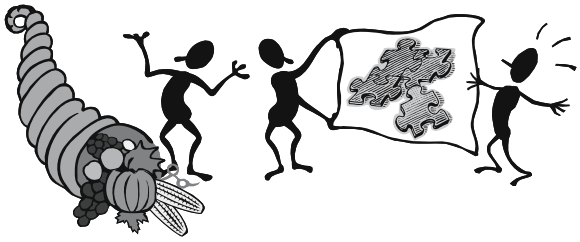
Große Softwarelösungen, mit denen oft das wirtschaftliche Funktionieren der sie einsetzenden Unternehmen vital verknüpft ist, werden über Jahre, manchmal über Jahrzehnte mit großem Aufwand gewartet und ausgebaut.

Abenteuer Systemwechsel

Unternehmen, die viel Geld, Wissen und Arbeit in eine bewährte Lösung investiert haben, setzen sich ungern dem Abenteuer aus, eine so maßgeschneiderte und gereifte Software durch ein neues System zu ersetzen. Sie zögern nicht selten auch dann, wenn aus technischen, organisatorischen und zuletzt auch aus wirtschaftlichen Gründen der Ersatz veralteter Systeme durch modernere Lösungen nahe liegt.

Das Ersetzen einer Software durch eine neue wird von Technikern und Anwendern unterschiedlich empfunden.

Bild 99: Neue Software kann man unterschiedlich empfinden.



Der Kern dieser Modernisierungsabneigung liegt zuweilen in der nicht unbegründeten Angst, dass sich mit dem neuen System die gewohnten Arbeitsgänge anders oder vielleicht schlechter als bisher erledigen lassen.

Die Anwender befürchten oft zu recht, dass sie ihre heute routinierten Tätigkeiten mit dem nächsten System mühsam neu erlernen müssen. Wenn auch das neue System so manches neue und nützliche kann: Möglich und in der Praxis schon vorgekommen ist, dass manch anderes, auf dessen Funktion die Anwender bei der täglichen Arbeit angewiesen sind, nicht mehr möglich sein wird.

13.1 Reengineering und Migration

Kein Reengineering- oder Migrationsprojekt und keine Entwicklung eines neuen Release beginnen mit dem Ziel, die oben beschriebene Situation herbeizuführen. Obwohl das neue System immer besser, schöner, schneller und leichter zu bedienen sein soll, stellt sich allzu oft heraus, dass sich die Anwender mit der Modernisierung ihrer täglich eingesetzten Software meist bestraft und nur selten geholfen fühlen.

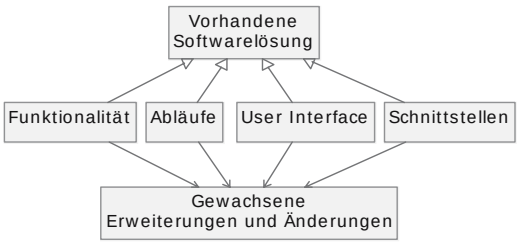


Bild 100: Etablierte Softwarelösungen sind oft organisch gewachsen

Der meiner Ansicht nach ausschlaggebende Grund für die Skepsis gegenüber Nachfolgesystemen ist: Die neue Software benutzt sich ganz anders.

Stellen Sie sich Ihr Auto nach Kundendienst vor: Da wo früher das Gaspedal war ist jetzt die Bremse, anstelle der Hupe ist der Blinker und da wo der Blinker war ist etwas, was sehr schön aussieht, aber dessen Funktion kein Kollege kennt. Muss das sein? Hätten Sie gerne, dass man Ihnen in Ihr Auto beim nächsten Servicetermin erst ein fünftes Rad einbaut und dass Sie dann noch den Führerschein neu machen müssen?

Das ist überspitzt formuliert, aber es könnte in etwa die Gefühlsvorstellung der Anwender über neue Softwarelösungen und Release-Wechsel treffen.

Beispiel: Funktional orientiertes Reengineering

Das folgende Beispiel zeigt Hinweise auf mögliche Ursachen des reservierten Anwenderempfindens bei Systemwechseln auf.

Die MoneyExperts Bank AG hat viel Geld in ihre Kreditsoftware investiert. Die auf die Belange der Bank angepasste Oberfläche hat sich bewährt. Die Anwendung arbeitet zuverlässig. Leider läuft die Software auf einem veralteten Betriebssystem, für das es kaum noch Support gibt. Neue technische Anforderungen, wie die Abwicklung von Kreditanfragen via Internet und die Verknüpfung mit einer Groupware zwingen die Bank schließlich zu einem Reengineering der Software auf einer aktuellen und zukunftssicheren technischen Plattform.

Die Bank beauftragt die interne IT zusammen mit der externen Firma KreditSoft mit der Planung und Umsetzung des Folgesystems. Der Auftrag beschreibt die technischen Anforderungen, die die Anwendung erfüllen soll, und auch, welche neuen Anwendungsfunktionen realisiert werden sollen. Der übrige Funktionsumfang soll in der neuen Technologie fachlich genauso, wie in der alten Anwendung umgesetzt werden. Die bewährte Bedienungsphilosophie soll im Wesentlichen beibehalten werden. Im Großen und Ganzen geht der Auftraggeber davon aus, dass die Funktionalität und die Bedienung der neuen Anwendung einfach vom Altsystem „abgeschrieben“ wird.

Natürlich soll die Oberfläche entsprechend dem Stand der Technik „modernisiert“ werden. So soll zum Beispiel die bisherige Eingabe von dreistelligen Kennzeichenkürzeln durch Auswahlfelder ersetzt werden. Auch sollen Listenansichten und parameterabhängige Workflows für verschiedene Kreditarten von berechtigten Anwendern frei konfiguriert werden können.

Die beauftragte KreditSoft stellt bald fest, dass das „Abschreiben“ vom Altsystem in die neue Technologie nicht so einfach ist, wie zunächst angenommen wurde. Insbesondere - erklärt KreditSoft dem Auftraggeber - sieht die Oberfläche in der neuen Technologie ganz anders aus, weil neue und komfortable Möglichkeiten der Bedienerführung zur Verfügung stehen. Das Abschreiben der Bedienungsweise vom Altsystem und stellenweise veraltete durch moderne Bedienelemente zu ersetzen, würde nichts bringen. Man müsse vielmehr die gesamte Oberfläche durchgängig neu planen.

Bei Entwurf der neuen Oberfläche werden die funktionalen Anforderungen zugrunde gelegt. Als Referenz für die Anforderungen an die Dialogmasken werden Ausdrucke von Masken aus dem Altsystem verwendet.

Die Oberfläche des Altsystems ist über Jahre ergänzt und optimiert worden. Eine Dokumentation aller Bedienungsmöglichkei-

ten gibt es nicht. Die Anwendungsexperten können im Tagesgeschäft nicht entbehrt werden.

Daher bleiben viele Eigenschaften des Altsystems - dies stellt sich später heraus - unberücksichtigt und werden erst nachträglich oder gar nicht in der neuen Software umgesetzt.

Die Begeisterung der Anwender für das neue System hält sich in Grenzen. Das Budget des Projekts wird erheblich überzogen. Die Zeitvorgaben ebenso.

13.2 UI-Beschreibung wieder verwenden

Wie hätte dieses in der Praxis leider nicht unübliche Szenario vermieden werden können? Wie kann man beim Reengineering das Weglassen entscheidender, bewährter Oberflächeneigenschaften vermeiden?

Wie schließlich kann man sicherstellen, dass bei zukünftigen Technologiewechseln die Oberfläche nicht neu entdeckt werden muss, sondern eine verbindliche Basis für die Umsetzung in neuer Technologie ist und bleibt?

Kontinuität bei UI-Entwicklung hilft, Bewährtes zu bewahren.

User Interface-Orientierung beim Reengineering hilft, Bewährtes zu bewahren. Das steigert die Akzeptanz der Innovation.

Herczeg unterstreicht die Wichtigkeit der Kontinuität beim Entwickeln und Weiterentwickeln von User Interfaces: „*Traditionelle Werkzeuge und Hilfsmittel wurden meist mit hohem Aufwand über lange Zeiträume für eine gute Bedienbarkeit entwickelt und dabei über viele Entwicklungsschritte an menschliche Physiologie, Fertigkeiten, Kenntnisse und Präferenzen optimal angepasst. Es haben sich auf diese Weise vielfältige, oft undokumentierte Gestaltungsprinzipien, Entwicklungs- und Testmethoden sowie nationale und internationale Standards entwickeln können. ... Bei computerbasierten Systemen fehlen diese langjährigen, evolutionären Entwicklungsprozesse und viele daraus abgeleiteten Erfahrungen auf Grund der vergleichsweise kurzen Historie interaktiver Computersysteme weitgehend*.“ [Herczeg 2005]

Bei Softwareoberflächen fehlen traditionelle, an der Bedienung und der Bedienbarkeit orientierte Bauweisen weitgehend und müssen erst mühsam gefunden werden. Einmal gefunden, sollten sie durch Übernahme der Oberfläche in die nächste technologische Umsetzungsplattform bewahrt bleiben.

Das UI-Modell ist nur insofern Bestandteil der Software, als dass es in der Software technisch umgesetzt wird. Ansonsten ist es eine von der Software unabhängige, virtuelle Verfahrens- und Verwendungsvorschrift.

Für den Anwender ist die Softwareoberfläche identisch mit dem Softwareprodukt. Er verlässt sich auf das Produkt und beurteilt es anhand der bewährten Eigenschaften der Oberfläche.

Die technologische Neuumsetzung wird aber von Entwicklern durchgeführt, die meist nicht mit der Altanwendung produktiv gearbeitet haben und wahrscheinlich auch nicht mit dem Folgesystem arbeiten werden. Daher wird die Bedeutung der Kontinuität in Verhalten und Bedienung der Anwendung von Entwicklern nur selten wahrgenommen. Für die Entwickler stehen naturgemäß die neue Technologie und deren Möglichkeiten im Vordergrund.

UI-Modelle liefern eine entscheidende Hilfestellung, um beide Sichten zu vereinen, und beantworten die oben formulierten Fragen. Ein UI-Modell ist ein technologie- und plattformunabhängiges Schema, in dem eindeutig beschrieben ist, wie die spezielle Bedienung, die Geschäftsabläufe des Unternehmens und die fachlichen Arbeitsschritte in der Bedienung einer Software abgebildet werden.

Generativer Ansatz schafft neue Möglichkeiten

Entscheidender Unterschied beim Entwickeln der Softwareoberfläche mit UI-Modellen ist: Die Präsentationsschicht wird nicht codiert oder mit einem Designwerkzeug gezeichnet, sondern aus den im Konzept modellierten Anforderungen generiert. Das ermöglicht:

- Plattformunabhängigkeit
- Flexibilität, Robustheit
- Qualitätserhöhung, Kostensenkung
- Produktivitätserhöhung

Anforderungen modellieren und Code aus Konzept generieren ist nachhaltiger als plattformspezifisches Design und Codierung. Die bewährte Oberfläche kann so den Lebenszyklus einer technischen Umsetzungsplattform überdauern und evolutionär mit neuen technischen Möglichkeiten weiterentwickelt werden.

13.3 Zusammenfassung und Ausblick

UI-Modelle haben einen vielfältigen Nutzen, der auch über Releases und Lebenszyklen einer Software hinweg wirkt. Der Einsatz von UI-Modellen zahlt sich in allen Projektphasen von der Anforderungsanalyse bis zur Realisierung und Test der Software aus.

Durch die Verwendung von UI-Modellen lassen sich Entwicklungskosten senken und Entwicklungsprozesse beschleunigen.

- Konzeptionsergebnisse können direkt in ablauffähige Anwendungsoberflächen umgewandelt werden.
- Die Dokumentation der Anforderungen ist strukturiert und nachvollziehbar.
- Entwicklungszyklen können iterativ und ohne Verlust von konzeptionellen Informationen erfolgen.
- Die Definition der Softwareoberfläche ist plattformunabhängig und überdauert Technologiewechsel.
- Wechselnde Anwenderforderungen werden toleriert. Das Softwareprodukt ist flexibler.
- Der Zeitraum zwischen der Änderung des Geschäftsprozesses und der Abbildung in den unterstützenden IT-Systemen wird verkleinert.

Die UI-orientierte Softwarearchitektur liefert:

- Ein praxisbezogenes ***Rahmenwerk*** für das Planen, das Spezifizieren, das Umsetzen und das Testen von Softwareoberflächen
- Eine auf das UI zentrierte ***Informationsarchitektur*** für das Entwickeln dialogintensiver Anwendungssoftware
- Eine ***Methodik*** für das eindeutige Beschreiben von Dialoginhalten und Verhalten von Softwareoberflächen
- Eine Grundlage für das frühzeitige ***Kommunizieren***, ***Abstimmen*** und für das systematische ***Testen*** von Anwendungsoberflächen

UI-Modelle helfen nicht nur beim Entwickeln der Oberfläche, sondern stellen auch Informationen für das Modellieren von Funktionen, Daten und technischen Abläufen zur Verfügung. User Interface-orientiertes Vorgehen unterstützt Kommunikation und Wissensmanagement im gesamten Projekt.

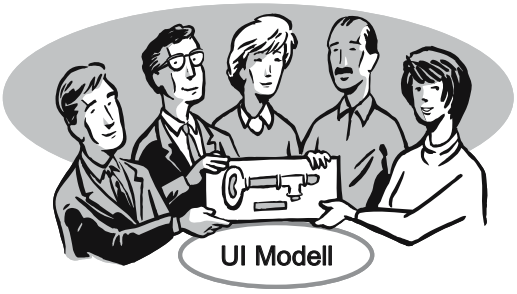


Bild 101: UI-orientierte Architektur unterstützt Knowledge Management

Ein User Interface-Modell enthält miteinander verknüpfte Informationen, die auf vielfältige Weise verwendet werden können.

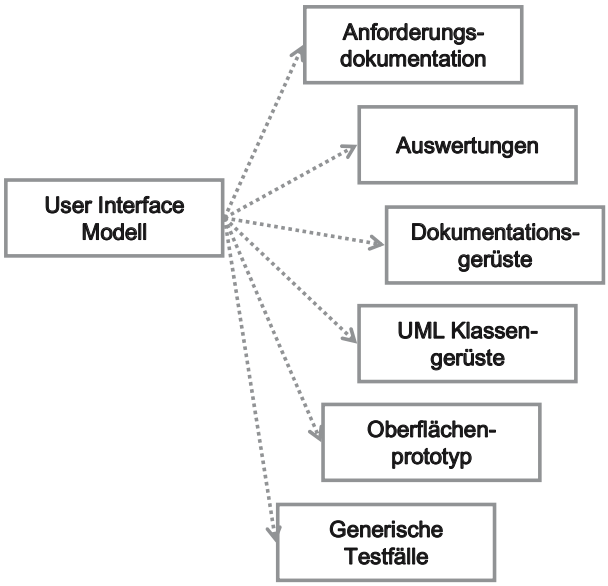


Bild 102: Möglichkeiten der Verwendung von UI-Modellen

Bis User Interface-Modelle flächendeckend eingesetzt werden, wird sicher noch eine umfangreiche Weiterentwicklung der heutigen Vorgehensweisen stattfinden.

Die notwendige Entwicklung betrifft sowohl unterstützende Werkzeuge und Methoden, als auch das Selbstverständnis der Anwendungsentwickler und die Nutzenwahrnehmung der Entscheider.

Die hier vorgestellte User Interface-Architektur hat einige Lücken und lässt Fragen offen. In der vorliegenden Fassung leistet sie keine erschöpfende Antwort auf alle Fragen der UI-Entwicklung, liefert jedoch einen praxistauglichen Überblick über die UI-Orientierung im Softwareentwicklungsprozess.

Auf der Grundlage der in diesem Buch aufgezeigten Ideen und Lösungsansätze können Sie viele neue Wege in der täglichen Arbeit an Ihren Softwareprojekten beschreiten und bessere Ergebnisse erzielen.

Die weitere Entwicklung bleibt spannend. Ich bin überzeugt davon, dass die User Interface-orientierte Architektur darin eine wesentliche Rolle spielt.

Vielen Dank, dass Sie dieses Buch gelesen haben.

Literaturverzeichnis

[Kidder 1982] Tracy Kidder: ***Die Seele einer neuen Maschine***; Birkhäuser Verlag, Basel Boston Stuttgart 1982

[Rose 1985] Frank Rose: ***Ins Herz des Verstandes***. *Auf der Suche nach der künstlichen Intelligenz*; Lev Roitman Verlag, München 1985

[Harel 1987] David Harel: ***Statecharts: A Visual Formalism for Complex Systems***; Science of Computer Programming 8, Elsevier 1987;

http://www.wisdom.weizmann.ac.il/~dharel/SCANNED.PAPERS/Statecharts.pdf

[Ebeling 1988] Adolf Ebeling: ***Gehirn, Sprache und Computer, Unerreichte Natur - künstliche Intelligenz***; Heinz Heise Verlag, Hannover 1988

[Myers 1992] Brad A. Myers (Hrsg.): ***Languages for developing user interfaces***; Jones and Barlett Publishers, Boston 1992

[Feldmann 1998] Clarence G. Feldmann: ***The Practical Guide to Business Process Reengineering Using IDEF0***; Dorset House Publishing, New York 1998

[Raskin 2001] Jef Raskin: ***Das intelligente Interface***; Adison-Wesley Verlag, ein Imprint der Pearson Education Deutschland, München 2001

[Lauer 2002] Michael Lauer: ***Python und GUI-Toolkits***; mitp-Verlag, Bonn 2002

[Shneiderm. 2002] Ben Shneiderman: ***User Interface Design*****, Deutsche Ausgabe**, 3. Auflage; mitp-Verlag, Bonn 2002

[Wessel 2002] Ivo Wessel: ***GUI-Design, Richtlinien zur Gestaltung ergonomischer Windows Applikationen***, *2. aktualisierte und überarbeitete Auflage*; Carl Hanser Verlag, München Wien 2002

[Blackwell 2003] Alan Blackwell, Thomas Green: ***Notational Systems - The Cognitive Dimensions of Notations Framework***; in John M.Caroll (Hrsg.): *HCI Models, Theories, and Frameworks Toward a Multidisciplinary Science* (103-133); Morgan Kaufmann Publishers, Elsevier Science, San Francisco 2003

[Ceri 2003] Stefano Ceri, Piero Fratenali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera: ***Designing Data-Intensive Web Applications***; Morgan Kaufmann Publishers, Elsevier Science, San Francisco 2003;

http://www.webml.org

[Jeckle 2004] Mario Jeckle, Chris Rupp, Jürgen Hahn, Barbara Zengler, Stefan Queins: ***UML 2 glasklar***; Carl Hanser Verlag, München Wien 2004;

http://www.uml-glasklar.com/

[Herczeg 2005] Michael Herczeg: ***Softwareergonomie***, *Grundlagen der Mensch-Computer-Kommunikation, 2. vollständig überarbeitete Auflage*; Oldenbourg Verlag, München Wien 2005

[Khazaeli 2005] Cyrus Dominik Khazaeli: ***Systemisches Design.*** *Intelligente Oberflächen für Information und Interaktion*; Rowohlt Verlag, Reinbek 2005

[Scharbert 2005] Karl Scharbert: ***Requirement Analysis realisieren***. *Praktischer Leitfaden für die Anforderungsanalyse bei IT-Projekten - Kundenanforderungen erfragen, verstehen und spezifizieren*; Vieweg Verlag, Wiesbaden 2005

Sachwortverzeichnis

A

Ablaufstruktur 62, 63, 68, 92, 94, 95, 100, 101, 102, 115, 124, 130, 136, 197
Anforderungen 8, 12, 13, 14, 16, 20, 26, 56, 57, 58, 61, 62, 64, 81, 83, 88, 101, 106, 112, 121, 124, 128, 130, 137, 138, 140, 142, 143, 147, 164, 168, 170, 171, 177, 179, 182, 191, 210, 211, 219, 220, 221, 226, 228, 229
Anforderungsperspektive 66, 107, 108, 137, 139, 147
Anwender ... 1, 2, 4, 8, 9, 11, 18, 24, 27, 29, 32, 36, 37, 38, 39, 42, 44, 45, 46, 47, 48, 50, 51, 52, 54, 60, 64, 66, 72, 92, 97, 102, 106, 112, 131, 132, 136, 137, 139, 144, 159, 162, 168, 178, 197, 205, 206, 207, 214, 223, 225, 227, 228
Architekt..... 57, 215, 220
Architektur .. 2, 10, 1, 5, 6, 7, 11, 13, 14, 15, 18, 25, 26, 32, 33, 46, 55, 90, 109, 113, 121, 123, 163, 164, 166, 167, 168, 170, 172, 173, 193, 214, 229, 230
Ausgabe ...42, 47, 52, 53, 91, 95, 136, 166, 170, 202, 231

B

Benutzeroberfläche . 9, 8, 10, 16, 27, 31, 59, 60, 62, 106, 121, 128, 131, 132, 133, 135, 140, 141, 143, 177, 209, 217
Browser..... 32, 224

D

Datenmodell 66, 135, 138, 210
Dialog...4, 29, 39, 45, 46, 48, 74, 93, 95, 96, 107, 114, 130, 136, 137, 147, 156
Dialogperspektive 66, 86, 87, 107, 108, 139, 147, 178

E

Eingabe 31, 47, 48, 52, 91, 158, 178, 179, 201, 226
Entwickler 2, 5, 11, 13, 18, 19, 21, 24, 26, 31, 33, 34, 37, 50, 55, 57, 60, 70, 81, 83, 87, 88, 101, 106, 138, 164, 172, 176, 182, 210, 211, 212, 214, 220, 221, 228
Erwartungen 21, 37, 214

F

funktionale Anwendung . 42, 175
Funktionalität ...9, 10, 19, 24, 32, 35, 42, 43, 46, 78, 80, 81, 101, 133, 167, 173, 212, 226

G

Generieren 63, 199
Guards..... 47, 134

I

Informationsarchitektur. 112, 229
Interaktionsperspektive... 66, 107, 108, 139, 147
Interpretieren 43, 47, 166

Iteration .. 102, 108, 109, 111, 112, 113, 114, 116, 117

J

Java 32, 60, 75, 78, 169

K

Kommunikation 5, 48, 65, 133, 138, 181, 229, 232
Kontextperspektive . 66, 108, 148, 179
Kontrollelemente . 22, 31, 33, 42, 50, 51, 52, 53, 54, 55, 56, 63, 71, 114, 119, 129, 140, 191, 201

L

Logik 22, 51, 54, 55, 56, 167, 218
logischer Schritt 96
LUCIA 18, 138, 139, 140, 141, 142, 143, 145, 146, 148, 149, 150, 158, 162, 193

M

Maschine 2, 9, 2, 5, 9, 27, 42, 45, 46, 47, 105, 132, 169, 231
Modellieren .. 61, 73, 96, 106, 122, 175, 176, 180, 185, 229
MVC .. 55, 165, 166, 167, 168, 169

N

Notation ... 21, 63, 64, 66, 67, 68, 72, 86, 87, 124, 125, 128, 129, 130, 136, 158
Notifikationen 55, 60, 72

O

Oberfläche .. 1, 3, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 27, 28, 31, 32, 33, 34, 37, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 54, 56, 57, 60, 71, 72, 81, 83, 84, 85, 86, 87, 88, 92, 95, 96, 97, 98, 101, 102, 105, 106, 107, 117, 119, 122, 124, 129, 131, 132, 133, 135, 137, 138, 139, 140, 142, 143, 144, 145, 146, 156, 162, 165, 167, 168, 170, 171, 173, 175, 176, 180, 181, 186, 187, 195, 197, 202, 203, 204, 205, 206, 207, 209, 210, 211, 212, 215, 216, 223, 226, 227, 228, 229

P

Präsentationslogik 36, 54, 59, 128, 129, 133, 146, 186
Praxis ... 6, 7, 14, 85, 225, 227
Programmierer 5, 87, 105, 122, 123, 138, 164, 215
Projektleiter 2, 5, 21, 81, 211, 216, 220, 221
Prototyp 18, 87, 172, 214
Prozesslogik 56, 148, 186, 187, 188, 189, 191, 193

R

Ressourcenperspektive .. 107, 108

S

Situationen 49, 113, 129, 133, 134, 140, 145, 148, 157, 186, 187, 188, 189, 190, 191, 192, 219, 224
Spezifikation 7, 9, 17, 18, 19, 20, 61, 62, 63, 72, 74, 75, 76, 79, 85, 86, 87, 88, 105, 106, 107, 108, 109, 116, 130, 134, 135, 136, 137, 142, 143, 146, 149, 165, 173, 177, 178, 195, 198, 199, 202, 206, 214, 219, 220

Spezifizierer 5, 24, 86, 87, 105, 122, 123, 138, 164, 175, 215, 218
State Charts 9, 51, 70, 72, 124, 125, 128, 129, 130, 131, 132, 133, 134
Strukturperspektive ... 66, 87, 107, 108, 139, 147, 178

T

Transaktion 24, 31, 33, 60
Transition ... 20, 79, 110, 113, 125, 126

U

UI-Modelle 18, 20, 84, 86, 101, 164, 167, 173, 175, 202, 209, 210, 221, 228, 229, 230
UI-Perspektiven ... 28, 60, 66, 107, 113, 119, 124, 129, 130, 137, 149
UML . 2, 9, 19, 20, 21, 22, 23, 24, 29, 60, 74, 75, 88, 89, 90, 91, 125, 167, 168, 169, 175, 178, 179, 180, 181, 183, 192, 232
Usability 2, 44, 123, 216
Use Case 3, 9, 10, 21, 23, 24, 88, 129, 176, 213
User Interface ..10, 1, 2, 3, 5, 6, 7, 8, 11, 13, 14, 15, 16, 17, 18, 19, 20, 24, 25, 26, 27, 34, 42, 46, 51, 54, 56, 57, 63, 64, 65, 70, 81, 83, 84, 87, 105, 109, 113, 116, 119, 125, 130, 134, 137, 138, 149, 163, 167, 170, 172, 176, 183, 191, 195, 196, 198, 201, 209, 210, 211, 212, 213, 214, 220, 223, 224, 229, 230, 231

V

Verhalten..44, 50, 51, 54, 64, 66, 75, 78, 85, 106, 122, 124, 129, 137, 218, 228, 229

W

Werkzeug...9, 57, 66, 69, 73, 106, 121, 170, 180, 189, 193
Widget.....52, 74, 78, 79, 80, 136, 137, 141, 142, 145, 146, 147, 148

X

XML 2, 29, 33, 63, 75, 77, 88, 139, 172, 173

Z

Zeichen 51, 143
Zielgruppe 1, 5, 15, 73